

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

Applicants: MERRIMAN, Dwight Allen et al.

Appl'n No.: 10/254,923

Filing Date: 26 September 2002

For: METHOD OF DELIVERY, TARGETING,  
AND MEASURING ADVERTISING  
OVER NETWORKS

Group Art Unit: 3627

Examiner: Harle, J.

**COMMISSIONER FOR PATENTS**

P.O. Box 1450  
Alexandria, VA 22313-1450

**DECLARATION UNDER 37 C.F.R. 1.131**

**OF DWIGHT A. MERRIMAN AND KEVIN J. O'CONNOR**

We, Dwight A. Merriman and Kevin J. O'Connor, declare that:

1. We are the named inventors of the claimed subject matter in the above identified patent application. We are informed that the application currently contains claims 23-44.
2. The invention as defined by the claims was completed by an actual reduction to practice prior to April 26, 1996. Evidence of this fact is shown by the following statements and the attached exhibit.
3. The actual reduction to practice included an affiliate node, an advertiser, a user node and an apparatus for advertising ("adserver"), as such terms are recited in the claims.
4. In particular, the system was tested prior to April 26, 1996 using a live affiliate Web site, <http://www.iaf.net>. The name of the IAF Web site is "Internet Address Finder", which Web site is active today.
5. To test the invented system, a link (an HTML tag) was inserted into a Web page at the IAF Web site at a position where an advertisement was to be displayed. Instead of displaying a stored banner advertisement or redirecting to an advertiser's Web site, the link at the IAF site redirected a user's browser to an adserver. The user node was implemented on a standard personal computer (PC) running a standard unmodified Internet browser.

6. An adserver was reduced to practice prior to April 26, 1996. The adserver was implemented as a live Internet node using standard PC hardware.
7. The adserver responded to an advertisement request from a user's browser based on the link from the affiliate node to select an advertisement in the form of a banner advertisement for display at the user's browser. Following click through by the user, the adserver redirected the user's browser to an advertiser. The advertiser was a standard Internet Web site.
8. The hardware for the adserver was a standard PC running the industry standard Windows NT operating system from Microsoft Corporation. The software for the adserver PC was written in the programming language C++. The portion of the C++ programming applicable to selection of advertising (the adserver function) is attached hereto as exhibit A.
9. Taking a closer look at exhibit A:
  - (a) the "GetRequest::service" method (Exhibit A, page DC 069492) shows that, depending upon the request from a user node, the adserver can respond by serving an ad to the user node via the "GetRequest::sendAd" method (Exhibit A, page DC 069494-95), or by enabling the user node to click through a served ad to the corresponding advertiser Web site via the "GetRequest::takeJump" method (Exhibit A, page DC 069495);
  - (b) in serving an ad via the "GetRequest::sendAd" method to the user node for display on the affiliate Web page:
    - i) the adserver retrieves from a database stored information about the user via the "User::lookupUser" method (Exhibit A, page DC 069499) and stored information about the affiliate Web page via the "SitePage::lookupPage" method (Exhibit A, page DC 069516);
    - ii) the stored user and page information is used to select an ad through the "Ad::getAd" method (Exhibit A, page DC 069503-04);
      - (1) the stored user and page information is used to match an ad's selection criteria in the "Ad::matches" method (Exhibit A, page DC 069502-03);
      - (2) the frequency of exposure of an ad at a user node is controlled in the "Ad::exposuresOK" method (Exhibit A, page DC 069502);

- iii) depending upon the nature of the selected ad, the adserver either retrieves the selected ad from a database and sends it to the user node via the "GetRequest::send" method (Exhibit A, page DC 069492-93), or the adserver identifies to the user node the ad's location at a different Web site, so that the user node may retrieve and display the ad.
10. The operation of each of the component nodes and the system combination of component nodes into a network of nodes was tested prior to April 26, 1996. The tests, which were witnessed prior to April 26, 1996, showed that each of the components and the system combination of components would work for its intended purpose.
11. Additional facts regarding the development of our invention and other background about the relevant technology may be found in our declarations under 37 C.F.R. 1.132, filed April 4, 2001 in Reissue Application No. 09/577,798, which are hereby incorporated by reference.
12. We, Dwight A. Merriman and Kevin J. O'Connor, individually declare under penalty of perjury that the above statements are true and correct to the best of our knowledge, information, and belief. We understand that willful false statements and the like are punishable by fine or imprisonment, or both (18 U.S.C. 1001) and may jeopardize the validity of any patent that issues from the above-identified patent application.

Respectfully submitted,

Date October 16, 2003

  
Dwight A. Merriman

Date October 17, 2003


  
Kevin J. O'Connor

Exhibit A

11-Jan-1996 13:25

REQUEST.H

getrequest.h

```

( !defined( GETREQUEST_H_ )
#define GETREQUEST_H_

#include "request.h"
#include "objects.h"

class GetRequest : public Request
{
public:
    GetRequest(Connection *C, Verb V,
                const char *requestText,
                const sockaddr_in &from) :
        Request(C, V, requestText, from) { }

    virtual void service();

protected:
    void whoAmI();
    void jumpingWhere(const char *from);
    void sendAck(const char *from);
    void activity(const char *activityStr); // Netscape 2.0 frames
    void sendFrame(const char *from);
    void takeJump(const char *from);
    void eyeState();

    void send(Database db, Ad *ad, User *u);

    // send info
    void sendInfo(const char *url);
    void el(const char *url);
}
endif

```

DX 50

HIGHLY  
CONFIDENTIAL

DC 069484

26-Sep-1995 12:39

MEMBERAD.H

// remembered.h

void rememberSend(Ad \*ad, User \*u, const char \*fromDoc);

// returns Ad ID  
DWORD queryAdSent(User \*u, const char \*fromDoc);

HIGHLY  
CONFIDENTIAL

DC 069485

23-Sep-1995 15:30

SERVER.N

// server.h

// General ad server startup stuff.

//

bool startServer();

HIGHLY  
CONFIDENTIAL

DC 069486

02-Jan-1996 14:24

```
STATUS.N
// status.h
void setStatus(const char *s);
extern int adSent;
extern int jumpsTaken;
extern int totalAdSendLatency;
extern int totalAdSendTime;
extern int timeOuts;
extern int poolTimeOuts;
extern int batter, lanDev, testAd;
void latencyWas(int n);
void adSendTimeWas(int n);
void adSent();
```

HIGHLY  
CONFIDENTIAL

DC 069487



03-Jan-1996 17:04

```

REQUEST.H
// request.h
//
// if defined(_REQUEST_H_)
//   define _REQUEST_H_
// include "/d/toolkit/sock.h"
enum Verb { UNKNOWN, GET, HEAD, POST };
class Connection;
class Request
{
public:
    Request(Connection *c, Verb v,
             const char *requestText,
             const sockaddr_in *from);
    virtual void service();
    DWORD getIp() const { return userip; }
    const char *getRequest() const { return request; }
    Connection *getConn() const { return c; }
    void sendInternalError();
protected:
    BOOL sendFile(const char *fileName, const char *insertStr = 0);
    Connection *c;
    const char *request;
    Verb v;
    CString fileName;
    DWORD userip;
};
void sendError(Connection *c, const char *msg, const char *headerField = 0);
#endif

```

HIGHLY  
CONFIDENTIAL

DC 069488

20-Dec-1995 17:131

HEADER.CPP

```

s = userAgent.Left(70);
message(s);
}

// derive information about the user from the request header
// void UserHeaderDerive(const char *requestHeader)
{
    const char *ua = strstr(requestHeader, "User-Agent:");
    if(ua == 0) {
        // if no user agent field, something weird we
        // don't know much about, don't assume unique.
        uniquenesses = unlikely;
    }
    else {
        ua++;
        while( *ua == ' ' )
            ua++;
        const char *p = strchr(ua, '\r');
        if(p) {
            CString userAgent(ua, p - ua);
            if( userAgent.Left(8) == "Mozilla/" ) {
                browser = "brNetscape";
                liftVer((const char *) userAgent + 8);
            }
            // OS
            liftOS(userAgent);
        }
        else if( userAgent.Left(12) == "NCSA Mosaic/" ) {
            browser = "brNCSA";
            liftVer((const char *) userAgent + 12);
        }

        // OS
        matchOS, userAgent, "Windows", osWin;
        matchOS, userAgent, "X11", osUnixUnknown;
        matchOS, userAgent, "X Window", osUnixUnknown;
    }
    else if( strcmp(userAgent, "iWENG/", 6) == 0 ) {
        browser = "brAOL";
        uniquenesses = unlikely;
        domainType = dtAOL;
        liftVer((const char *) userAgent + 6);
        os = osWin;
    }
    else if( strcmp(userAgent, "nolbrowser/", 10) == 0 ) {
        browser = "brAOL";
        uniquenesses = unlikely;
        domainType = dtAOL;
        liftVer((const char *) userAgent + 11);
        os = osMac;
    }
    else if( userAgent.Left(28) == "Microsoft Internet Explorer/" ) {
        // Microsoft Internet Explorer/4.40
        browser = "brMicrosoft";
        liftVer((const char *) userAgent + 28);
        os = osWin32;
        matchOS, userAgent, "Windows 95", osWin95;
    }
    else if( userAgent.Left(8) == "HotJava/" ) {
        browser = "brHotJava";
        liftVer((const char *) userAgent + 8);
    }
    else if( userAgent.Left(16) == "Enhanced_Mosaic/" ) {
        browser = "brEnhancedMosaic";
        liftVer((const char *) userAgent + 16);
        os = osWin;
        if( userAgent.Find("Win32") >= 0 )
            os = osWin32;
    }
    else if( userAgent.Left(11) == "NetCruiser/" ) {
        liftVer((const char *) userAgent + 11);
        browser = "brNetCruiser";
        os = osWin;
    }
}

```

Page 1 (3)

20-Dec-1995 17:131

HEADER.CPP

```

// header.cpp
//
#include "stdafx.h"
#include "objects.h"
#include "d/toolkit/inf_util.h"

const char *cBrowser[] = {"User-Agent:"};

void message(const char *);

bool User::check(CString& userAgent, const char *pat, Browser b, OS o)
{
    if( browser != brUnknown )
        return FALSE;

    int l = strlen(pat);
    if( userAgent.Left(l) == pat ) {
        browser = b;
        os = o;
        const char *p = userAgent;
        p++;
        p = strchr(p, '/');
        if(p) {
            liftVer(p + 1);
        }
        return TRUE;
    }
    return FALSE;
}

static void match(OS o, const char *userAgent, const char *pat, OS o)
{
    if( strstr(userAgent, pat) != 0 )
        os = o;
}

void User::liftOS(const CString& userAgent)
{
    if( userAgent.Find("X11") >= 0 ) {
        os = osUnixOther;
        matchOS, userAgent, "SunOS", osUnixSun;
        matchOS, userAgent, "hp-UX", osUnixHP;
        matchOS, userAgent, "Linux", osUnixLinux;
        matchOS, userAgent, "OSF", osUnixOSF;
        matchOS, userAgent, "AIX", osUnixAIX;
        matchOS, userAgent, "IRIX", osUnixIRIX;
    }
    else if( userAgent.Find("Windows") >= 0 ) {
        if( userAgent.Find("32bit") >= 0 ||
            userAgent.Find("95") >= 0 )
            os = osWin32;
        else {
            os = osWin16;
        }
    }
    else if( userAgent.Find("Win95") >= 0 ) {
        os = osWin95;
    }
    else if( userAgent.Find("Win16") >= 0 ) {
        os = osWin16;
    }
    else if( userAgent.Find("Macintosh") >= 0 ) {
        os = osMac;
        matchOS, userAgent, "ppc", osMacPPC;
        matchOS, userAgent, "68K", osMac68;
    }
    else if( userAgent.Find("WinNT") >= 0 ) {
        os = osWinNT;
    }
    else {

```

DC 069489

HIGHLY  
CONFIDENTIAL

```

else {
    check(userAgent, "OmniWeb", brOmniWeb, osNEXT);
    check(userAgent, "lynx", brLynx, osUnknown);
    check(userAgent, "IBM WebExplorer", brWebExplorer, osOS2);
    check(userAgent, "AIR Mosaic", brAIRMosaic, osWin);
    check(userAgent, "SPY Mosaic", brSPYMosaic, osWin);
    check(userAgent, "MacWeb", brMacWeb, osMac);
    check(userAgent, "NetChange", brChameleon, osWin);
    check(userAgent, "NetSurf", brNetSurfer, osNEXT);
    check(userAgent, "CHNWorks", brCNN, osWin);
    check(userAgent, "InternetHot", brHotNet, osUnknown);
    check(userAgent, "Emulasy", brEmulasy, osUnknown);
    check(userAgent, "PipeMacWeb", brPipeMacWeb, osMac);
    check(userAgent, "InternetICI", brICI, osUnknown);
    check(userAgent, "Quartardeck", brQuartardeck, osUnknown);
    check(userAgent, "MCSA Mosaic for the X", brMCSA, osUnixUnknown);
    if (check(userAgent, "Gworldbrowser", brGWorld, osMac)) {
        if (userAgent.Find("68K") >= 0)
            os = osMac68;
        else if (userAgent.Find("ppc") >= 0)
            os = osMacPPC;
        uniqueness = uNo;
        domainType = dtEWorld;
    }
    else if (check(userAgent, "PRODIGY", brProdigy, osUnknown)) {
        uniqueness = uNo;
        domainType = dtProdigy;
    }
    else if (check(userAgent, "Delphi", brDelphi, osUnknown)) {
        uniqueness = uNo;
        domainType = dtDelphi;
    }
    else if (browser == brUnknown) {
        TRACE("unknown userAgent: %s\n", (const char *) userAgent);
        if (osFOS(userAgent));
    }
}

if (userAgent.Find("via proxy") >= 0) {
    proxy = TRUE;
    if (!uniqueness == uUnknown)
        uniqueness = uNo;
}
}

```

**HIGHLY  
CONFIDENTIAL**

DC 069490

23-Dec-1995 11:01

```

LOCATION.CPP
// location.cpp
#include "stdafx.h"
#include "object.h"
#include "d/toolkit/mapdate.h"
#include "d/toolkit/tzutil.h"

// next line should be in tzutil.h
extern CountryTimezoneMap mapCountryTimeZones;

struct IsDaylightSavings
{
    IsDaylightSavings()
    {
        TIME_ZONE_INFORMATION t;
        DWORD r = GetTimeZoneInformation(&t);
        daylightSavings = r == TIME_ZONE_ID_DAYLIGHT;
    }
    BOOL daylightSavings;
} Isd;

tm* Location::userRelativeTime( time_t timeRelative )
{
    int utc_offset;
    int daylight_bias;

    if( country == 256 ) {
        if( !getStaticTimezoneInfo(&state, utc_offset, daylight_bias) )
            return FALSE;
    }
    else if( country == 0 ) {
        return FALSE;
    }
    else {
        DWORD dwBias;
        if( !mapCountryTimeZones.Lookup(country, dwBias) )
            return FALSE;
        utc_offset = LOWORD(dwBias);
        daylight_bias = HIWORD(dwBias);
    }

    time_t tTime;

    // if timeRelative == 0, this assumes that they want the time
    // relative to the current time
    tTime = timeRelative;
    if( !tTime )
        tTime = time(&tTime);

    if( Isd.daylightSavings && daylight_bias != TZ_BIAS_UNDETERMINED )
        tTime += daylight_bias * 60 * 60;
    else
        tTime += utc_offset * 60 * 60;
    return gmtime(&tTime);
}

```

HIGHLY  
CONFIDENTIAL

DC 069491

18-Jan-1996 17:12

GETREQUEST.CPP

```

// getrequest.cpp
//
#include "atdafx.h"
#include "stream.h"
#include "d/cookie/sock.h"
#include "getrequest.h"
#include "remembered.h"
#include "d/cookie/iaf_util.h"
#include "log.h"
#include "status.h"
#include "d/cookie/crit.h"
#include "d/cookie/dbutil.h"
#include "d/cookie/dbpool.h"

extern CriticalSection fast;
//extern Database iaMain;

extern ostream errLog;
extern int activity;

extern const char *browserNames();

const char *progName = "AdSvr";

void message(const char *);

void recalcsi();

DWORD startLatency, endLatency;

// This used to prevent multiple concurrent FTP
// requests right now because our FTPD implementation
// only does one at a time.
//
extern HANDLE fphMutex;

void GetRequest::service()
{
    const char *p = strchr(request, ' ');
    if (p)
    {
        fileName = CString(request, p - request);
        else
        {
            fileName = request;
        }
        if (fileName.Left(4) == "/ad/" )
        {
            sendad(const char *) (fileName + 4);
        }
        else if (fileName.Left(9) == "/adframe/" )
        {
            sendFrame(const char *) (fileName + 9);
        }
        else if (fileName.Left(6) == "/jump/" )
        {
            takeJump(const char *) (fileName + 6);
        }
        else if (fileName.Left(10) == "/activity/" )
        {
            activity(const char *) (fileName + 10);
        }
        else if (fileName.Left(7) == "/whoami/" )
        {
            whoami();
        }
        else if (fileName.Left(8) == "/viewad/" )
        {
            CString szFileName;
            szFileName.Format("c:/lan/ads/ta", (LPCTSTR)fileName+8);
            sendFile(szFileName);
        }
        else if (fileName.Left(11) == "/stats.htm?" )
        {
            sendError("404 Not Found: Result (forecast moved to another server)");
        }
        //stats(const char *) (fileName + 11);
        else if (fileName.Left(10) == "/sendinfo/" )
        {
            sendInfo(const char *) (fileName + 10);
            return;
        }
        else if (fileName.Left(4) == "/si/" )
        {
            // send info stuff
            si(const char *) (fileName + 4);
        }
    }
}

```

HIGHLY  
CONFIDENTIAL

DC 069492

18-Jan-1996 17:12

GETREQUEST.CPP

```

}
else if (fileName.Left(9) == "/sysstate" ) {
    sysState();
}
else {
    const char *p = fileName;
    if (strnicmp(p, "/java/", 6) == 0 ) {
        if (strchr(p, "...") == 0 )
            sendFile(p);
        else
            sendError("404 Not Found");
    }
    else {
        if ( *p == '/' )
            p++;
        if ( *p == 0 ) {
            // send default
            sendFile("c:/lan/html/default.htm");
            return;
        }
        else {
            if (strchr(p, '/') == 0 && strchr(p, '\\') == 0 &&
                strchr(p, "...") == 0 )
            {
                CString f = "c:/lan/html\\";
                f += p;
                sendFile(f);
                return;
            }
            sendError("404 Not Found");
        }
    }
}

// Normally we adjust SI for an ad as it is delivered.
// However, occasionally should do all ads in case one hasn't
// been delivered but time has passed.
static int counter; // adjust constant as traffic increases
if ( ++counter > 200 ) {
    counter = 0;
    Crit c(fast);
    if ( AllFree() ) { // recal SI for all ads
        recalcsi();
    }
    else {
        counter = 175; // try again soon
    }
}

const char cHeader[] =
"http/1.0 200 OK\r\nContent-Type: image/gif\r\nContent-Length: ";

// send() should commit the DN if it does any DN operations because
// the caller commits ahead of time so that the transaction won't
// remain open while the file is sent.
//
void GetRequest::send(Database db, Ad *ad, User *u)
{
    CString hdr = cHeader;
    const BUFSIZE = 32000;
    char buf[BUFSIZE];
    Cookie sendCookie;
    if ( !db ) {
        if ( !u->hasCookie() ) {
            // If a user record already exists, it's probably because
            // this IP address is shared with other users (proxy, IP pool,
            // etc.) So, we want to create another record; we don't want
            // to assign the same cookie to different people!
            u->userID = 0; // create new record
        }
        // generate a cookie for the user
    }
}

```

```

text << "<html><System State/></html></body></html>"
text << "<table border=1 cellpadding=3>"
text << "<tr><td><b>Name/b></td><td><b>Type/b></td><td><b>SI</td></tr>\n";
text << "<tr><td><b>Sent/b></td><td><b>Ads Booked</b></td></tr>\n";
text << "<td><b>Ads Sent</b></td><td><b>Ads Booked</b></td></tr>\n";

// Get a db connection to lock the ads array so that
// it isn't reloaded or anything while we are processing.
Database *db = getFromPool();

for( int i = 0; i < ads.GetSize(); i++ ) {
    ad *ad = ads.GetAt(i);
    text << "<tr><td><a href='http://ad.lanargeta.com/viewad/'>"
        ad->fileName.MakeLower();
    text << "<td><a href='http://ad.lanargeta.com/viewad/'>"
        ad->fileName.MakeLower();
    text << "<td><a href='http://ad.lanargeta.com/viewad/'>"
        ad->fileName.MakeLower();
    text << "<td><a href='http://ad.lanargeta.com/viewad/'>"
        ad->fileName.MakeLower();
    text << "<td><a href='http://ad.lanargeta.com/viewad/'>"
        ad->fileName.MakeLower();
    text << "<td><a href='http://ad.lanargeta.com/viewad/'>"
        ad->fileName.MakeLower();
    text << "<td><a href='http://ad.lanargeta.com/viewad/'>"
        ad->fileName.MakeLower();
}

releaseToPool(db);

text << "<table>"
text << "</table>"
text << "</body></html>";

int n = text.pcount();
char temp[100];
itoa(n, temp, 10); // content length
hdr += temp;
hdr += "\r\n\r\n";

c->write( (const char *) hdr, hdr.GetLength() );
c->write(buf, n);

// diagnostic
old GetRequest::whoAmI()

Database *db = "getFromPool()";

User *user = User::lookupUser(db, userIP, request);
user->lookupAncillaryInfo(db);

CString hdr = "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nContent-Length: ";
buf = 0;
stringstream text(buf, 32000, ios::out);

// fill content
text << "<html><body bgcolor=#ffffff><hl><h1></h1></html>"
text << "<pre>";
user->describe(db, text);
text << "</pre></body></html>";

int n = text.pcount();
char temp[100];
itoa(n, temp, 10); // content length
hdr += temp;
hdr += "\r\n\r\n";

c->write( (const char *) hdr, hdr.GetLength() );
c->write(buf, n);

delete user;
releaseToPool(db);

}

// diagnostic
void GetRequest::jumpingWhere(const char *from)
{
    ASSERT(FALSE);
    // fix for multi-db conns

    User *user = User::lookupUser(userIP, request, FALSE);
}

```

```

u-shasCookie = TRUE;
u-shasPermanent(db);
sendCookie.value = u->getId();
}

// release DB here so that we don't keep a db connection occupied
// while sending the ad
db.commit();
releaseTopool((db);
}

CFile f;
int n = 0;
if( v == GET ) {
    CString s = ad->fullName();
    if( !f.Open(s, CFile::modeRead | CFile::shareDenyWrite) ) {
        message( CString["couldn't open "] + s );
        TRACE["couldn't open %s\n", (const char *) s);
        ASSERT(FALSE);
        return;
    }
}
n = f.Read(buf, BUFSIZE);
ASSERT( n != 0 && n != BUFSIZE );

}
else {
    n = GetFileSize( ad->fullName() );
    // next line is a test for MCSA Mosaic HEAD
    //n = 1;
}

char temp[100];
loadn, temp, 10); // content length
hdr = temp;
if( !sendCookie.isNull() ) {
    wprintf(temp,
        "\r\nSet-Cookie: IAF=1; path=/; expires=Wed, 09-Nov-99 23:59:00 GMT";
        sendCookie.value);
    hdr += temp;
}

// last-modified time
hdr += "\r\nLast-Modified: " + curHTTPTime();

//test
// hdr += "\r\nPragma: no-cache";
hdr += "\r\n\r\n";

endlatency = GetTickCount();

c->write( (const char *) hdr, hdr.GetLength() );
if( v == GET )
    c->write(buf, n);
}

// diagnostic
void GetRequest::getState()
{
    static char *typeStr[] = {
        "Normal",
        "Test",
        "Bart",
        "lan Dev"
    };

    CString hdr =
        "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nContent-Length: ";
    char buf[32000];
    buf = 0;
    ostream text(buf, 32000, ios::out);

    // fill content
    text << "<body bgcolor=ffffff>\r\n";
}

```

**HIGHLY  
CONFIDENTIAL**

DC 069493

**NETREQUEST.CPP**

```

type = InfoRequest;
break;
case 's':
type = Sale;
break;
default:
OK = FALSE;
}

if (ok) {
const char *p = activate;
if (p != '/')
OK = FALSE;
else {
p++;
const char *q = strchr(p, ' ');
if (q == 0)
OK = FALSE;
else
sitekey = CString(
).
}

if (ok) {
Database *db = getFrom(
User *user = User::lookUp(
DHOPD advertiserID =
// todo: fix if not a
if (user->userID != 0)
{
Cursor c(*db);
c.bind(SQL_C_LONG,
char sql(1024) =
advertiserId, site
c.exec(sql);
OK = c.fetchNext()
)
}

db->commit();

if (ok) {
activity++;
if advertiserID
logActivity(user,
)

delete user;
releaseToPool(db);
}

if (ok) {
message! CString("in
CString fact
/ sendError(c, "404 Not
)

void GetRequest::sendAddCo
if (from && strcmp(fr
from == 4;

Database *db = getFrom(
static DHOPD lastFTP;
startLatency = GetTickD
User *user;
SitePage *page;
Ad *ad;

user = User::lookUpUser
if (db == 0) {
page = 0;
}
}

```

**STANDARD.CPP**

```

Ad *ad = Ad::findSentTo(user, from);

if( ad == 0 ) { // fix
    delete user;
    return;
}

SitePage *page = SitePage::lookupPage(from, request);

CString hdr =
    "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nPragma: no-cache\r\nContent-Length: ";
char buf(32000);
if( ad == 0 )
    atStreamText(buf, 32000, los, out);

// fill content
text << "html><body><h1>Jump Redirect</h1>";
text << "pre>";
text << "jumping from document: " << from << "\r\n";
text << "jumping would jump to: "
text << "a href=\"" << (const char *) ad->jumpTo << "\">";
text << "a href=\"" << (const char *) ad->jumpTo << "</a>\r\n\r\n";
text << (const char *) ad->jumpTo << "</a>\r\n\r\n";
text << (const char *) ad->jumpTo << "</a>\r\n\r\n";
text << (const char *) ad->jumpTo << "</a>\r\n\r\n";

CString fn = ad->fileName;
text << "<center><img src=\"" <<
    << (const char *) fn
    << "\">";
text << "</pre></body></html>";

int n = text.pcount();
char temp[100];
for(int temp, 10; // content length
    hdr << temp;
    hdr << "\r\n\r\n";
    c->write( (const char *) hdr, hdr.GetLength());
    c->write(buf, n);
    logJump(ad, user, page);
    delete page;
    delete ad;
    delete user;

void GetRequest::sendFrame(const char *from)
{
    CString a = "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\n";
    a << "html><body><center><a href=\"" << http://206.4.219.5/jump/" <<
    a << from;
    a << "</body></html>";
    a << "\><center></html>"; // width=468 Height=60
    a << "\></a></center></html>";
    c->write( (const char *) a, a.GetLength());
}

void GetRequest::activity(const char *activityStr)
{
    // go ahead and send for best response time
    sendFile("c:\\lan\\html\\dot.gif");

    BOOL bad = FALSE;

    // send the file first
    ActivityType type;
    CString skey;
    BOOL ok = TRUE;
    switch( *activityStr )
    {
        case 'a':
            type = Interest;
            break;
        case 'i':
    }
}

```

**HIGHLY  
CONFIDENTIAL**

**DC 069494**

```

else {
    page = SitePage::lookupPage(-db, from, request);
}
ad = Ad::getAd(-db, user, page, v == GET);

// TRACE("get %s\n", from);
// }

static int randCutoff = 0; //RAND_MAX / 4;

bool doFTP = user->tempUserObject() &&
user->ftpTried && user->uniqueness > unlikely && user->proxy &&
rand() < randCutoff && (startLatency - lastFTP > 6000);
DWRND dw;
if( doFTP )
    dw = WaitForSingleObject(&ftpMutex, 0);
if( doFTP && dw != WAIT_FAILED && dw != WAIT_TIMEOUT ) {
    lastFTP = startLatency;

    // Remember that we're doing FTP for user. Only do once.
    user->ftpTried = TRUE;
    user->updateFtpTried("-dw");

    // Redirect
    CString a = "Location: ";
    a += "ftp://206.4.219.6/";
    char buf[10];
    sprintf(buf, "%ix", user->getID());
    a += buf;
    a += '\n';
    CString fn = ad->getFileName();
    a += (const char *) fn;

    errLog << "Trying FTP\n";
    errLog << "user = " << user->getID() << '\n';
    errLog << "browser = " << browserNames[(int) user->browser] << '\n';
    errLog << "url = " << a << '\n';

    a += "\r\n";
    sendError(c, "302 Moved Temporarily", a);

    VERIFY( ReleaseMutex(&ftpMutex) );

    logAdSend(ad, user, page);

    errLog.Flush();

    db->commit();
    releaseToPool(db);
}
else {
    //t.cs.leave();
    send(-db, ad, user); // this function calls releaseToPool()
    //t.cs.enter();
    if( v == GET ) {
        static int counter;
        if( ++counter & 2 )
            ad->calcSI(); // update SI every 4 or so deliveries

        rememberSend(ad, user, from);
        logAdSend(ad, user, page);
        if( user->timedOut ) {
            if( db == 0 )
                poolTimeOuts++;
            else
                timeOuts++;
        }
        // state
        c->close(); // flush send
        DWRND endSend = GetTickCount();
        lastLatency = startLatency;
    }
}

```

```

        adSendTimeVar(endSend - startLatency);
    }
}

// delete ad;
delete page;
delete user;
}

void GetRequest::takeJump(const char * _from)
{
    Database &db = *getFromPool();

    // jumpingWhere(from);
    // return;

    User *user = User::lookupUser(db, userIP, request, FALSE);

    if (from && strlen(cmp1_from, "www.", 4) == 0)
        _from += 4;

    CString from;

    {
        const char *p = strchr(_from, '?');
        if (p == 0) {
            from = _from;
            char buf[512];
            sprintf(buf, "no ismap %d.%s", user == 0 ? 999 : (int) user->browser, (const char *)
                message(buf);
        }
        else
            from = CString(_from, p - _from);
    }

    Ad *ad = Ad::findSentTo(user, from);
    SitePage *page = SitePage::lookupPage(db, from, request);

    // (l.cs.leave());
    CString s = "Location: ";
    s += ad->jumpTo; // + "?from=1a*";
    s += "\r\n";
    sendError(C, "301 Moved Permanently", s);
    c->close();
    // (l.cs.enter());

    // Must do this so activity will be logged properly.
    // See GetRequest::activity().
    user->makePermanent(db);

    logJump(ad, user, page);

    delete page;
    delete ad;
    delete user;
    db.commit();
    releaseToPool(&db);
}

```

**HIGHLY  
CONFIDENTIAL**

**DC 069495**



16-Jan-1996 18:10

OBJECTS.CPP

```

"Genio".
0,0,0,0,0,0,
"Reserved for ISP Names"
};

const char *ISPNames[] = {
    "ISP",
    "NetCom",
    "PSI",
    "UNet",
    "Advantis",
    "Concentric Research Corp.",
    "CRL",
    "MCI",
    "Portal Information Network"
};

const char *salesStr[] = {
    "unknown",
    "$1 - $49,999",
    "$50,000 - $99,999",
    "$100,000 - $249,999",
    "$250,000 - $499,999",
    "$500,000 - $999,999",
    "$1 million - $4,999,999",
    "$5 million - $9,999,999",
    "$10 million - $49,999,999",
    "$50 million - $99,999,999",
    "$500 million - $999,999,999",
    "$1 billion and over"
};

const char *empStr[] = {
    "unknown",
    "1 - 4",
    "5 - 9",
    "10 - 14",
    "15 - 19",
    "20 - 49",
    "50 - 99",
    "100 - 499",
    "500 - 999",
    "1,000 and over"
};

const char *genderStr[] = {
    "unknown",
    "Male",
    "Female"
};

const char *timesStr[] = {
    "12am-1am",
    "1am-2am",
    "2am-3am",
    "3am-4am",
    "4am-5am",
    "5am-6am",
    "6am-7am",
    "7am-8am",
    "8am-9am",
    "9am-10am",
    "10am-11am",
    "11am-12pm",
    "12pm-1pm",
    "1pm-2pm",
    "2pm-3pm",
    "3pm-4pm",
    "4pm-5pm",
    "5pm-6pm",
    "6pm-7pm",
    "7pm-8pm",
    "8pm-9pm",
    "9pm-10pm"
};

```

Page 1(9)

16-Jan-1996 18:10

OBJECTS.CPP

// objects.cpp

#include "stdafx.h"

//-----

```

const char *uniqueNames[] = {
    "Unknown", "No", "Unlikely", "Likely", "Yes"
};

```

```

const char *browserNames[] = {

```

```

    "Unknown",
    "Netscape",
    "Mosaic",
    "MCSA Mosaic",
    "AOL Browser",
    "HotJava",
    "Microsoft",
    "OmniWeb",
    "Lynx",
    "NetCruiser",
    "IBM WebExplorer",
    "AIR Mosaic/Spry Mosaic",
    "Macheb",
    "NetManage Chameleon",
    "Netsurfer",
    "Enhanced Mosaic",
    "eWorld Browser",
    "Prodigy Browser",
    "Delphi Browser",
    "CNN Browser",
    "InterNotes",
    "Mollagong/ATM Emularray",
    "PipMacWeb",
    "InternetMCI",
    "Quarterdeck Mosaic"
};

```

```

const char *osNames[] = {

```

```

    "Unknown",
    "Win16",
    "Win32",
    "Windows",
    "NT",
    "WinNT",
    "OS/2",
    "Macintosh",
    "Mac 68k",
    "Mac PowerPC",
    "Unix (brand unknown)",
    "Unix (other)",
    "Unix (Sun)",
    "Unix (Linux)",
    "Unix (HP)",
    "Unix (AIX)",
    "Unix (OSF)",
    "Unix (IRIX)",
    "NEXT",
    "Unix (SGI)"
};

```

```

const char *domainTypeNames[] = {

```

```

    "Unknown",
    "Commercial", "Education", "Government",
    "Military", "K-12", "Foreign", "Networks",
    "Organisations"
};

```

0.

```

    "AOL",
    "Prodigy",
    "CompuServe",
    "Delphi",
    "eWorld",
    "MSN",
    "DowJones"
};

```

HIGHLY  
CONFIDENTIAL

DC 069496

16-Jan-1996 16:10

## OBJECTS.CPP

```

return userID;
}

User::User()
{
    timedOut = FALSE;
    userID = 0;
    uniqueness = uUnknown;
    ip = 0;
    browser = brUnknown;
    bVer1 = bVer2 = 0;
    os = osUnknown;
    domainType = dtUnknown;
    // for(int i = 0; i < MAXSICS; i++)
    // sicCodes[i] = 0;
    nEmployees = 0;
    salesVolume = 0;
    proxy = FALSE;
    isNetWorkDescription = FALSE;
    ftpTried = FALSE;
    hasCookie = FALSE;
}

void User::describe(Database& db, ostream& text)
{
    in_addr ipAddr = (in_addr) ip;
    text << "<b>ip: " << ip << "</b>"
    << (int) ipAddr.S_un.S_un_b.b1 << " ";
    << (int) ipAddr.S_un.S_un_b.b2 << " ";
    << (int) ipAddr.S_un.S_un_b.b3 << " ";
    << (int) ipAddr.S_un.S_un_b.b4 << "</b>";

    gender = " ";
    if( !emailAddr.isEmpty() ) {
        // get name/gender
        Cursor c(db);
        CString g,f,l;
        c.bind(g);
        c.bind(f);
        c.bind(l);
        Signature s;
        s.email = emailAddr;
        s.pullEmail();
        if( !s.l.domain.isEmpty() ) {
            char sql[1024] = "select gender, name, iname1 from listings where emailname=";
            addValue(sql, s.l.emailName, FALSE);
            strcat(sql, " and domain=");
            addValue(sql, s.l.domain, FALSE);
            c.exec(sql);
            if( c.fetchNext() ) {
                if( g.GetLength() )
                    gender = g.GetAt(0);
                name = { ' ', ' ', ' ', ' ' };
                cap(name);
            }
        }
        db.commit();
    }

    text << "<b>unique: " << uniqueness << "</b>"
    << (hasCookie ? "Yes" : "No") << "</b>";
    text << "<b>cookie: " << browserName << "</b>"
    << (int) bVer1 << " " << (int) bVer2 << "</b>";
    text << "<b>browser ver: " << browserName << "</b>"
    << osName << "</b>";
    text << "<b>domain type: " << domainType << "</b>";
    text << "<b>proxy: " << proxy << "</b>";
    text << "<b>name: " << name << "</b>";
    text << "<b>title: " << title << "</b>";
    text << "<b>state: " << locationState << "</b>";
    text << "<b>zip code: " << locationZipCode << "</b>";
    text << "<b>area code: " << locationAreaCode << "</b>";
    text << "<b>phone: " << phone << "</b>";
    text << "<b>e-mail: " << emailAddr << "</b>";
}

```

16-Jan-1996 16:10

## OBJECTS.CPP

```

"10pm-11pm",
"11pm-12am"
);

const char *dowStr[] = {
    "Sunday",
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday",
    ""
};

if( !defined(_JUSTSTRINGS) )
{
    #include <stream.h>
    #include <fstream.h>
    #include <winsock.h>
    #include <objects.h>
    #include <tables.h>
    #include <d/toolkit/inf_util.h>
    #include <d/toolkit/db.h>
    #include <d/toolkit/dbutil.h>
    #include <d/derive/sqlderive.h>
    #include <d/newderive/sig.h>
    #include <rememberad.h>

    extern ostream errLog;
    extern Ad *badKeyErrorAd;

    int nextAd = 0;

    #if !defined(_DERIVE)
    int mnde();
    #endif
    extern Ad *defaultAd;

    //-----
    // User
    //-----

    BOOL User::cookieCapable() const
    {
        // todo: add new version of Internet Explorer

        return browser == brNetscape &&
            (bVer2 >= 1 ||
             bVer1 > 1);
    }

    void User::deriveDomainTypeFromBrowser()
    {
        switch( browser ) {
            case braOL:
                domainType = dtAOL;
                break;
            case brEWorld:
                domainType = dtEWorld;
                break;
            case brProdigy:
                domainType = dtProdigy;
                break;
            case brDelphi:
                domainType = dtDelphi;
                break;
            default:
                ;
        }
    }

    DNDP User::getID() const
    {

```

HIGHLY  
CONFIDENTIAL

DC 069497

16-Jan-1996 18:10

## OBJECTS.CPP

```

CString desc;
Cursor cldb;
c.bind(SQL_C_LONG, slevel, 4);
c.bind(category);
c.bind(desc);
char sql[512];
wsprintf(sql, "select interest_level, category, name from interests, user_interests\
where interests_id=interest_id and user_id=11d\
order by interest_level DESC", userID);
c.exec(sql);
while( c.fetchNext() ) {
    char buf[128];
    wsprintf(buf, "%11d ", slevel);
    text << buf;
    text << category << " " << desc << "\r\n";
}
db.commit();
}

void User::getNetworkInfo(Database db, BOOL *timedOut)
{
    if( !p == 0 ) {
        ASSERT(FALSE);
        return;
    }

    // if( domainType != dtUnknown ) {
    //     // got dt from header info
    //     // if ISP/OSP, location and sales, etc. don't apply.
    //     // if we have done a trace, location does apply.
    //     // for ISP/OSP.
    //     // if domainType != dtNetcom // did trace for netcom
    //     return;
    // }

    // Note: do the following for all domain types to at least get country.
    NetworkNumber n;
    n = JustNetworkNumber(ip);

    char buf[256] =
        "select domain_type, sales, num_employees, sic, country, state, zipcode, areacode from networks\
Cursor cldb;
if( domainType < dtAOL ) {
    c.bind(SQL_C_LONG, domainType, sizeof(domainType));
    c.bind(SQL_C_LONG, salesVolume, sizeof(salesVolume));
    c.bind(SQL_C_LONG, numEmployees, sizeof(numEmployees));
    cCodes.bind(c);
} else {
    strcpy(buf, "select country, state, zipcode, areacode from networks where netnumber=");
    strcat(buf, n.sqlStr());
    c.bind(SQL_C_LONG, slocation, country, sizeof(location.country));
    c.bind(location.state);
    c.bind(location.zipCode);
    c.bind(SQL_C_LONG, slocation, areacode, sizeof(location.areaCode));
    if( timedOut == 0 )
        c.setTimeOut(1);
    c.exec(buf);
    if( c.timedOut() )
        *timedOut = TRUE;
    else
        c.fetchNext();
    if( uniqueness == uUnknown && (int) domainType >= (int) dtAOL )
        uniqueness = uUnlikely;
    if( domainType >= dtAOL ) {
        salesVolume = 0;
        numEmployees = 0;
        sicCodes.makeNull();
        if( domainType != dtNetcom && domainType != dtISPother ) {

```

Page 5(9)

16-Jan-1996 18:10

## OBJECTS.CPP

```

text << "<b>location: </b>";
if( location.country == 256 ) {
    text << "US";
} else {
    text << "country # " << location.country;
}
text << "\r\n";

text << "<b>job function:</b>" << "\r\n";
text << "<b>gender:</b>";
if( gender == 'm' || gender == 'n' )
    text << "Male";
else if( gender == 'f' || gender == 'g' )
    text << "Female";
else
    text << "?";
text << "\r\n";
text << "<b>";
Domain *d = Domain::lookupDomain(ip);
if( d == 0 ) {
    text << "isNo company information available.</b>";
} else {
    text << "<b>domain name: </b>" << (const char *) d->domain << "\r\n";
    text << "<b>bus. name: </b>" << (const char *) d->name << "\r\n";
    text << "<b>address: </b>" << (const char *) d->address[0] << "\r\n";
    for( int i = 1; i < MAXADDR; i++ ) {
        if( !d->address[i].isEmpty() ) {
            text << " " << (const char *) d->address[i] << "\r\n";
        }
    }
    text << "<b>contact: </b>" << (const char *) d->contact[0] << "\r\n";
    for( i = 1; i < MAXCONTACT; i++ ) {
        if( !d->contact[i].isEmpty() ) {
            text << " " << (const char *) d->contact[i] << "\r\n";
        }
    }
    text << "<b>industries: </b>" << "\r\n";
    {
        sicCodes.reset();
        SICCode sc;
        while( sicCodes.getNext(sc) ) {
            text << " " << sc.asTextPadded() << "\r\n";
        }
    }
    for( i = 0; i < MAXSICS; i++ )
        if( d->sicCodes[i] )
            text << d->sicCodes[i] << " ";
    text << "\r\n";
    if( numEmployees )
        text << "employees: ";
    else
        text << "less than 25 (unknown)";
    text << "\r\n";
    text << "<b>revenue: </b>";
    if( salesVolume )
        text << salesVolume;
    else
        text << "less than $1MM (unknown)";
    delete d;
}

text << "<b>";
text << "<b>You are interested in the following:</b>" << "\r\n\r\n";
text << "Interest Level Category: Description\r\n";
text << "-----\r\n";
{
    DWORD level;
    CString category;

```

HIGHLY  
CONFIDENTIAL

DC 069498

16-Jan-1996 18:10

OBJECT3.CPP

```

}
else {
    // lookup by cookie
    u = lookupUserID(db, cookie.value, tmount);
    if (u) {
        u->uniqueness = uYes;
        u->ip = ip;
    }
    else {
        if (defaultAdMode) {
            // db conn down
            u = new User;
            u->uniqueness = uYes;
            u->ip = ip;
            u->userIn = cookie.value;
        }
        else {
            // Couldn't find user record, we will need to
            // assign a new cookie. Do not load by IP, because
            // we don't want this user sharing a record
            // with others without cookies.
            // Note: generally, this shouldn't happen.
            cookie.value = 0;
        }
    }
}
else if ( !_timedOut ) {
    u = lookupUserByAddress(db, ip, tmount);
    if (u) {
        u->ip = ip;
        u->hasCookie = FALSE;
    }
}
if (u == 0) {
    // make a default user object
    u = new User;
    //u->uniqueness = uNo;
    u->ip = ip;
    u->timedOut = _timedOut;
}
u->headerDerive(requestHdr);
if ( !cookie.isNull() )
    u->hasCookie = TRUE;
if ( loadDemographics && !_timedOut )
    u->getNetworkInfo(db, realTime ? &u->timedOut : 0);
return u;
}
//-----
// SitePage
Ad* Ad::findSentTo(User *user, const char *fromDoc)
{
    DWORD adNum = queryAdSent(user, fromDoc);
    for ( int i = 0; i < adNum; i++ ) {
        Ad* ad = *ads.GetAt(i);
        if ( ad.id == adNum )
            return new Ad(ad);
    }
    if ( badKeyErrorAd && adNum == badKeyErrorAd->id )
        return badKeyErrorAd;
    if ( user->uniqueness == unlikely ) {
        if ( defined(errLog) )
            errLog << "findSentTo failed uniqueness-likely\n";
        errLog << "user = " << user->userID << "\n";
        errLog << "from doc = " << fromDoc << "\n";
    }
}

```

Page 7(3)

16-Jan-1996 18:10

OBJECT3.CPP

```

// don't know location, except country
location.state.Empty();
location.zipCode.Empty();
location.areaCode = 0;
}
else {
    sicCodes.checkNull();
}
if ( defined(_DERIVE) )
    const char cCookie[] = "Cookie:";
void User::liftVer(const char *verStr)
{
    int v1 = 0, v2 = 0;
    sscanf(verStr,
        "%d.%d", &v1, &v2);
    bVer1 = v1;
    bVer2 = v2;
}
/*
 * User::lookupUserID(DWORD userID)
 */
User * User::lookupUserID(DWORD userID)
{
    DWORD userID = networkNodeTable.getUserID(ip, FALSE);
    if ( userID == 0 ) {
        // Try to get domain info at least. Note: if user is uniquely
        // identifiable, derive data process will create a record for the
        // user as soon as it gets a chance.
        userID = networkNodeTable.getUserID(justNetworkNumber(ip), TRUE);
    }
    if ( userID ) {
        return lookupUserByID(userID);
    }
    return 0;
}
//
extern defaultAdMode;
User* User::lookupUser(Database db, DWORD ip, const char *requestHdr, BOOL loadDemographics,
{
    BOOL _timedOut = &db == 0;
    BOOL _tmout = realTime ? !_timedOut : 0;
    //-----
    // get cookie for lookup
    Cookie cookie;
    const char *ck = strstr(requestHdr, cCookie);
    if ( ck )
        cookie.getFromHeader(ck, "AP");
    //-----
    // lookup
    User *u = 0;
    if ( !cookie.isNull() ) {
        if ( !_timedOut ) {
            u = new User;
            u->uniqueness = uYes;
            u->ip = ip;
            u->userID = cookie.value;
            u->timedOut = TRUE;
        }
    }
}

```

HIGHLY  
CONFIDENTIAL

DC 069499

16-Jan-1996 18:10

OBJECT9.CPP

```
    errLog.Flush();  
    SendIf  
    {  
        // temp: just return first Ad (ISS)  
        //return new Ad( ads.ElementAt(0) );  
        return new Ad( "defaultAd" );  
    }  
    // return 0;  
    SendIf  
    SendIf
```

**HIGHLY  
CONFIDENTIAL****DC 069500**

11-Oct-1995 10:21

```

COOKIE.CPP
// cookie.cpp
//
#include "stdafx.h"
#include "objects.h"
//-----
// Cookie
const Cookie& Cookie::operator=(const char *s)
{
    secant(s, "tix", svalue);
    return *this;
}

/*static*/
Cookie Cookie::alloc(DWORD userID)
{
    ASSERT(userID != 0);
    Cookie k;
    k.value = userID;
    return k;
}

// Get value for a particular cookie name from the HTTP header
// hdr - points to the Cookie field in the header
void Cookie::getFromHeader(const char *hdr, const char *name)
{
    hdr += 7; // skip "Cookie:"
    const char *p = strchr(hdr, '\r');
    if(p) {
        CString nm = name;
        nm += ".";
        const char *q = strstr(hdr, nm);
        if(q && q < p)
            *this = q + nm.GetLength();
    }
}

```

HIGHLY  
CONFIDENTIAL

DC 069501



18-Jan-1996 15:15

MATCH.CPP

```
// SIC
if( nsicCodes ) {
    BOOL ok = FALSE;
    int i = 0;
    while( i ) {
        if( i >= nsicCodes ) {
            // no match
            return FALSE;
        }
        SICCodes pattern = sicCodes[i];
        user->sicCodes.reset();
        SICCode sc;
        while( user->sicCodes.getNext(sc) ) {
            if( pattern.matches(sc) ) {
                ok = TRUE;
                break;
            }
        }
    }
    // Site and page categories
    // Do last, because this is expensive (disk hit)
    if( siteCategories.isEmpty() ) {
        BOOL v;
        if( sitepage == 0 )
            return FALSE;
        sitepage->loadCategories();
        for( int i = 0; i < sitepage->categories.GetSize(); i++ )
            if( siteCategories.Lookup(sitepage->categories.GetAt(i), v) )
                return TRUE;
        return FALSE;
    }
    return TRUE;
}

inline BOOL Ad::criteriaOK(Database& db, User *user, SitePage *page)
{
    return spreadOK(page) &&
        (!isTargeted()) &&
        (matches(user, page) && exposuresOK(db, user));
}

// todo: if reload ads, need to handle the fact that
// one may still be in use and can't just delete.
// (crit sect released during sending of file.)
// Ad::Ad::getAd(Database& db, User *user, SitePage *page, BOOL increment)
{
    const SIMAX = 1000000;
    if( user->uniqueness < unlikely )
        return defaultAd;
    if( page == 0 ) {
        if( badKey/ErrorHandler )
            return badKey/ErrorHandler;
        ASSEPT(FALSE);
    }
    if( increment )
        nextAd = (nextAd + 1) % nAds();
    int lowestSI;
    Ad *adLowestSI;
    const int start = nextAd;
    // Do a test ad, if appropriate. Always do these first so that
```

Page 3 (6)

18-Jan-1996 15:15

MATCH.CPP

```
if( (o & os) == 0 )
    return FALSE;

// Browser
o = 1 << ((int) user->browser);
if( (o & browser) == 0 )
    return FALSE;

// DomainType
int userISP = 0;
int dt = (int) user->domainType;
if( dt >= (int) dtISPOther ) {
    userISP = dt - (int) dtISPOther + 1;
    dt = 0;
}

// ISP
o = 1 << userISP;
if( (o & isp) == 0 )
    return FALSE;

else {
    o = 1 << dt;
    if( (o & domainType) == 0 )
        return FALSE;
}

// location
if( locations != 0 ) { // if ISP, don't know location (yet)
    if( userISP )
        return FALSE;
    BOOL ok = FALSE;
    for( int i = 0; i < nLocations; i++ ) {
        if( user->location.in(locations[i]) ) {
            ok = TRUE;
            break;
        }
    }
    if( !ok )
        return FALSE;
}

// hour of day / day of week
if( hoursOfDay != 0xffff || daysOfWeek != 0x7f ) {
    tm *t;
    if( isAbsoluteTime() ) {
        time_t now;
        time(&now);
        t = localtime(&now);
    }
    else {
        t = user->location.userRelativeTime();
        if( t == 0 )
            return FALSE;
    }
    if( (hoursOfDay & (1 << t->tm_hour)) == 0 )
        return FALSE;
    if( (daysOfWeek & (1 << t->tm_wday)) == 0 )
        return FALSE;
}

// sales
if( salesVolume != 0x7fffffff ) {
    o = 1 << user->salesVolume;
    if( (o & salesVolume) == 0 )
        return FALSE;
}

// # employees
if( nEmployees != 0x7fffffff ) {
    o = 1 << user->nEmployees;
    if( (o & nEmployees) == 0 )
        return FALSE;
}
```

HIGHLY

CONFIDENTIAL

DC 069503



```

// a truly random distribution is used for them rather than
// leftovers.
static int testCounter;
if( testCounter % 4 == 0 ) { // just try every 4 to save CPU
    test ad avail?
    lowestSI = 1051;
    int i = start;
    while( 1 ) {
        Ad ad = *ads.GetAt(i);
        if( ad.type == Test && ad.si < lowestSI && ad.criteriaOK(db, user, page) ) {
            lowestSI = ad.si;
            adLowestSI = &ad;
        }
        i = (i + 1) % nAds();
        if( i == start )
            break;
    }
    if( lowestSI <= 1050 )
        return adLowestSI;
}

lowestSI = SIMAX;
adLowestSI = defaultAd;

// Check remnants first. This way, we don't
// have to do ad matching for any targeted ads
// with high SI's.
int i = start;
while( 1 ) {
    Ad ad = *ads.GetAt(i);
    if( ad.type == Normal && !ad.isTargeted() && ad.si < lowestSI && ad.spreadOK(page) ) {
        lowestSI = ad.si;
        adLowestSI = &ad;
    }
    i = (i + 1) % nAds();
    if( i == start )
        break;
}

// this is temp; eventual all placements will have book rates
// you'll want to remove this to get better performance (no ad matching
// if remnant has worst SI).
static int counter = 1;
// for ads with no booking amount,
// allow a targeted ad to run sometimes
if( lowestSI == 1100 )
    lowestSI++;

// for ads where we don't care about # impressions,
// bias in favor of targeted
if( lowestSI == 1100 )
    lowestSI++;

// todo later: if ads are sorted by si (lowest first),
// you can quit matching as soon as you find
// one. Could be a good optimization.

// do targeted
i = start;
while( 1 ) {
    Ad ad = *ads.GetAt(i);
    if( ad.type == Normal && ad.isTargeted() &&
        ad.si < lowestSI &&
        ad.spreadOK(page) &&
        ad.matches(user, page) &&
        ad.exposuresOK(db, user) ) {
        // found a good one
        lowestSI = ad.si;
    }
}

```

```

adLowestSI = &ad;
}
i = (i + 1) % nAds();
if( i == start )
    break;
}

if( lowestSI > 1400 ) {
    // do either a better ad or an fan dev ad
    static int counter;
    if( counter % 5 == 0 ) {
        // do an fan dev ad
        i = start;
        while( 1 ) {
            Ad ad = *ads.GetAt(i);
            if( ad.type == fanDev && ad.criteriaOK(db, user, page) ) {
                // found a good one
                adLowestSI = &ad;
                break;
            }
            i = (i + 1) % nAds();
            if( i == start )
                break;
        }
    }
    else {
        // do better
        lowestSI = SIMAX;
        i = start;
        while( 1 ) {
            Ad ad = *ads.GetAt(i);
            if( ad.type == Better &&
                ad.si < lowestSI &&
                ad.criteriaOK(db, user, page) ) {
                // found a good one
                adLowestSI = &ad;
                lowestSI = ad.si;
            }
            i = (i + 1) % nAds();
            if( i == start )
                break;
        }
    }
}

return adLowestSI;
}

```

```

REQUEST.CPP
// request.cpp
//
#include "stdafx.h"
#include "d/toolkit/sock.h"
#include "request.h"
#include "d/toolkit/inf_util.h"

if defined(_CONSOLE)
#include <stream.h>
endif

if defined(_IAP)
extern ostream *outlog;
void impression();
endif

extern CString gratuitous;

Request::Request(
    Connection *c,
    Verb *v,
    const char *request,
    const sockaddr_in from,
    c_l_c, request(request), v(v)
)
{
    userip = from.sin_addr.s_addr;
}

int spider = 0;

BOOL Request::readFile(const char *fileName, const char *insertStr)
{
    if defined(_IAP)
        outlog << "-end << fileName << ' << inet_ntoa( (in_addr) userip ) << '\n';
    endif

    const char insertChar = '-';
    BOOL isSpider = FALSE;
    CString hdr = "HTTP/1.0 200 OK\r\nContent-Type: ";
    if (strchr(fileName, ".class") != 0) {
        hdr += "application/java\r\nContent-Length: ";
    }
    else if (strchr(fileName, ".gif") != 0) {
        hdr += "image/gif\r\nContent-Length: ";
    }
    else {
        hdr += "text/html\r\nContent-Length: ";
    }
    if defined(_IAP)
        impression();
    endif

    int gnt = 0;
    if (strchr(request, "-Agent: Lycos") != 0)
        gnt = 1;
    if (strchr(request, "InfoSeek Robot") != 0)
        gnt = 2;
    if (strchr(request, "-Agent: WebCrawl") != 0)
        gnt = 3;

    if (gnt)
    {
        isSpider = TRUE;
        spider++;
        if defined(_CONSOLE)
            cout << "***** Robot << gnt << " << "*****\n";
        endif
    }

    const BUFSIZE = 13800;
    char buf(BUFSIZE + 20);
    CFile f;
    CFileException fe;

```

HIGHLY  
CONFIDENTIAL

DC 069505

```

REQUEST.CPP
if (v == GET || v == POST) {
    if (if.Open(fileName, CFile::modeRead | CFile::shareDenyWrite, &fe) ) {
        if (fe.m_cause == CFileException::accessDenied) {
            sendError(c, "404 Not Found (Access Denied)");
        }
        else if (fe.m_cause == CFileException::sharingViolation) {
            sendError(c, "404 Not Found (Sharing Violation)");
        }
        else
            sendError(c, "404 Not Found");
        return FALSE;
    }
    n = f.Read(buf, BUFSIZE);
    if (n == 0) {
        sendError(c, "404 Not Found");
        return FALSE;
    }
    ASSERT( n != 0 && n != BUFSIZE );

    char *p = buf;
    if (insertStr) {
        while(1) {
            p = strchr(p, insertChar);
            if (p == 0)
                break;
            int l = strlen(insertStr);
            memmove(p + 1, p + 1, strlen(p + 1));
            memcpy(p, insertStr, l);
            p += l;
            n += l;
        }
    }

    if (isSpider) {
        if (gratuitous.IsEmpty()) {
            if defined(_CONSOLE)
                cout << "gratuitous empty. (?)\n";
            endif
        }
        else {
            buf[n] = 0;
            char *p = strstr(buf, "</BODY>");
            if (p) {
                for (int i = 0; i < 20; i++) {
                    strcpy(p, gratuitous);
                    p += gratuitous.GetLength();
                }
                strcpy(p, "</BODY><<HTML>");
                n = (p - buf) + 14;
            }
            else {
                if defined(_CONSOLE)
                    cout << "/body?\n";
                endif
            }
        }
    }

    char templ[100];
    itoa(n, templ, 10); // content length
    hdr += templ;
    hdr += "\r\n\r\n";
    c->write( (const char *) hdr, hdr.GetLength() );
    if (v == GET || v == POST)
        c->write(buf, n);
    return TRUE;
}

```

03-Jan-1996 15:53

REQUEST.CPP

```

void Request::service()
{
    const char *p = strchr(request, ' ');
    if (p)
        fileName = CString(request, p - request);
    else
        fileName = request;

    {
        const char *p = fileName;
        if (*p == '/')
            p++;
        if (*p == 0)
            // send default
            //sendFile("k:\\my documents\\internet address findex\\lafmain.htm");
        if defined(IAP)
            sendFile("c:\\laf\\html\\lafmain.htm");
        return;
    }
    else {
        if (strchr(p, '\\') == 0 && strchr(p, "...") == 0) {
            if (strchr(p, '/') != 0) {
                CString f = "c:\\lan\\";
                f += p;
                sendFile(f);
                return;
            }
            else
            {
                if defined(IAP)
                    CString f = "c:\\laf\\html\\";
                if defined(MANAGE)
                    CString f = "c:\\lan\\manage\\";
                else
                    ASSERT(FALSE);
                CString f = "jskldt";
                //CString f = "k:\\my documents\\ad federation\\";
                f += p;
                sendFile(f);
                return;
            }
        }
        sendError(c, "404 Not Found");
    }

    void Request::sendInternalError()
    {
        sendError(c, "500 Internal Server Error");
    }
}

```

HIGHLY  
CONFIDENTIAL

DC 069506

```

REMEMBERAD.CPP
// rememberad.cpp
#include "stdafx.h"
#include "objects.h"
#include "rememberad.h"
#include "d/toolkit/hashwf.h"
#include "d/toolkit/crit.h"

const SZ = 107313;

// this is a test
static int cr;
#define INCRIT ( ASSERT(cr==0); cr++; )
#define OUTCRIT ( ASSERT(cr==1); cr--; )

void message(const char *);

extern CriticalSection fast;

struct Key
{
    DWORD userID;
    DWORD fromHash;

    BOOL operator==(const Key& k) const
    {
        return userID == k.userID && fromHash == k.fromHash;
    }
}

void setID(User *u)
{
    if (u->userID)
        userID = u->userID;
    else
        userID = u->slp;
}

void setFrom(const char *from)
{
    fromHash = hashwf(from);
}

UINT HashKey(Key key)
{
    return key.userID * key.fromHash;
    // default identity hash - works for most primitive values
    // return ((UINT)(void*)(DWORD)key) >> 4;
}

struct Value
{
    DWORD adSent;
    DWORD time;
};

class Memory
{
public:
    Memory() : sent(100)
    {
        sent.InitHashTable(SZ);
    }

    void remember(Keys k, DWORD adID);
    DWORD lookup(Key& k);

private:
    void purge();
    ChapKey, Keys, Value, Values sent;
    Memory;
} // todo: fix

```

HIGHLY  
CONFIDENTIAL  
DC 069507

```

REMEMBERAD.CPP
// todo: nonunique hashes
//
//DWORD hash(const char *from, User *u)
//{
//    char buf[10];
//    sprintf(buf, "%lx", u->getID());
//    CString a = buf;
//    a += from;
//    return hashwf(a);
//}

void Memory::remember(Keys k, DWORD adID)
{
    static int count;
    if (++count > 1000 ) {
        count = 0;
        purge();
    }

    Value v;
    v.adSent = adID;
    v.time = ::GetTickCount();
    sent.SetAt(k, v);
}

DWORD Memory::lookup(Key& k)
{
    Value value;
    if (sent.Lookup(k, value) ) {
        return value.adSent;
    }
    return 0;
}

void Memory::purge()
{
    const LIMIT = 1000 * 60 * 60 * 24; // too much?

    if (sent.GetCount() > SZ ) {
        message("remember map > SZ");
    }

    DWORD now = ::GetTickCount();
    POSITION p = sent.GetStartPosition();
    while (p) {
        Key k;
        Value v;
        sent.GetNextAssoc(p, k, v);
        if (now - v.time > LIMIT)
            sent.RemoveKey(k);
    }
}

void rememberSend(Ad *ad, User *u, const char *fromDoc)
{
    Crit c(fast);
    // INCRIT
    Key k;
    k.setID(u);
    k.setFrom(fromDoc);
    memory.remember(k, ad->id);
    // OUTCRIT
}

DWORD queryAdSent(User *u, const char *fromDoc)
{
    Crit c(fast);
    // INCRIT
    Key k;
    k.setID(u);
    k.setFrom(fromDoc);
    DWORD d = memory.lookup(k);
    // OUTCRIT
    return d;
}

```

19-Jan-1996 10:15

SOLDB.CPP

```

{
    Crit c(fast);
    if( allFree() ) {
        for( int i = 0; i < ads.GetSize(); i++ ) {
            delete ads.GetAt(i);
            ads.RemoveAt(i);
            defaultAd = 0;
            ok = loadAds(ads, 0, TRUE, TRUE, FALSE, FALSE);
        }
        break;
    }
    Sleep(50);
}

if( ok )
    message("Ad reload completed OK");
else
    message("Ad reload (allure!)");

// note: this isn't getting called yet
void closeSOLDB()
{
    lafmain.close();
}

//-----
// Ads

AdArray ads;

class AdCursor : public Cursor
{
public:
    AdCursor()
    {
        bind(SOL_C_LONG, ad.id, 4);
        bind(SOL_C_LONG, ad.type, sizeof(ad.type));
        bind(SOL_C_LONG, ad.os, sizeof(ad.os));
        bind(SOL_C_LONG, ad.browset, sizeof(ad.browset));
        bind(SOL_C_LONG, ad.domainType, sizeof(ad.domainType));
        bind(SOL_C_LONG, ad.iap, sizeof(ad.iap));
        bind(ad.fileName);
        bind(ad.jumpTo);
        bind(SOL_C_LONG, ad.frequency, sizeof(ad.frequency));
        bind(SOL_C_LONG, ad.imageSeries, sizeof(ad.imageSeries));
        bind(SOL_C_LONG, ad.maxImpressions, sizeof(ad.maxImpressions));
        bind(SOL_C_LONG, ad.nShown, sizeof(ad.nShown));
        bind(SOL_C_LONG, ad.startTime, sizeof(ad.nShown));
        bind(SOL_C_LONG, ad.endTime, sizeof(ad.flags));
        bind(SOL_C_LONG, ad.flags, sizeof(ad.flags));
        bind(SOL_C_LONG, ad.hoursOfDay, sizeof(ad.hoursOfDay));
        bind(SOL_C_LONG, ad.daysOfWeek, sizeof(ad.daysOfWeek));
        bind(SOL_C_LONG, ad.nEmployees, sizeof(ad.nEmployees));
        bind(SOL_C_LONG, ad.saleVolume, sizeof(ad.saleVolume));
        bind(SOL_C_LONG, ad.active, sizeof(ad.active));
        bind(ad.description);
        bind(SOL_C_LONG, ad.maxAmount, sizeof(ad.maxAmount));
        bind(ad.sxPONumber);
        bind(SOL_C_LONG, ad.approved, sizeof(ad.approved));
        bind(SOL_C_LONG, ad.nJumps, sizeof(ad.nJumps));
    }
}

Ad ad;
};

```

```

// ... TODO!!! This function is not thread-safe.
void recalcs()
{
    for( int i = 0; i < ads.GetSize(); i++ ) {
        ads.ad = ads.GetAt(i);
        ad.calcSI();
    }
}

```

Page 1 (8)

19-Jan-1996 10:15

SOLDB.CPP

```

// sqldb.cpp
//
#include "stdafx.h"
#include "stream.h"
#include "objects.h"
#include "d/oolkit/db.h"
#include "d/oolkit/inf_util.h"
#include "d/oolkit/dbutil.h"
#include "d/oolkit/dbpool.h"
#include "d/oolkit/crit.h"

// this ad displayed if a bad sitekey is encountered
const cbadkeyAdID = 49;

extern CriticalSection fast;

Database lafmain;
void message(const char *);
BOOL defaultAdMode = FALSE;

static int ucctOfs;
static void localToUCT(time_t & t)
{
    t += ucctOfs;
}

// This is temporary. Used for non-unique users.
// Eventually will be smarter about what to send to
// these users.
Ad *defaultAd = 0;

Ad *badKeyErrorAd = 0;

typedef CherryAd * Ad * AdArray;

BOOL loadAds(AdArray & ads,
             DWORD advertiserID, // 0=all
             BOOL forTargeting, // if forTargeting, update Ad.targetSites to reflect
                               // site exclusions
             BOOL activeOnly, // active only
             BOOL includeExpired, // include where enddate has past or where all delivered
             BOOL newestFirst, // order from newest to oldest
             DWORD siteID = 0);

BOOL openSOLDB()
{
    lafmain.open();
    openDBPool();
}

if( !lafmain.open() )
    return FALSE;

if( !openDBPool() )
    return FALSE;

if( !lafmain.Open(0, FALSE, FALSE,
                  "ODBC/DSN=inf(UID=sa;PWD=sa",
                  "FALSE"/ TRUE) )
    return FALSE;

if( !loadAds(ads, 0, TRUE, TRUE, FALSE, FALSE) )
    return FALSE;

return TRUE;

void reloadAds()
{
    BOOL ok = FALSE;
    message("waiting to reload ads...");
    AfxGetApp()->m_pMainWnd->invalidate();
    AfxGetApp()->m_pMainWnd->UpdateWindow();
    while( 1 ) {

```

HIGHLY  
CONFIDENTIAL

DC 069508

19-Jan-1996 10:15

SQLDB.CPP

```

        where = TRUE;
        strcat(sql, " where");
    }
    strcat(sql, " active=");
    addValue(sql, active, FALSE);
}

if( advertiserID ) {
    if( where ) {
        strcat(sql, " and");
    } else {
        where = TRUE;
        strcat(sql, " where");
    }
    strcat(sql, " advertiser=");
    addValue(sql, advertiserID, FALSE);
}

if( approvesiteID ) {
    if( where ) {
        strcat(sql, " and");
    } else {
        where = TRUE;
        strcat(sql, " where");
    }
    strcat(sql, " not exists (select * from approved where site_id=");
    addValue(sql, approvesiteID, FALSE);
    strcat(sql, " and ad_id=");
}

if( newestFirst ) {
    strcat(sql, " order by id desc");
}

rs.exec(sql);

while( 1 ) {
    // defaults in case null
    rs.ad.flags = 0;

    if( rs.fetchNext() )
        break;

    // if for debug, don't load. You can make this test a registry
    // setting if you like so that you can load debug records, or
    // add a cmd line setting.
    if( rs.ad.isProduction() )
        continue;

    if( rs.isNull(12) ) {
        time_t now;
        rs.ad.starttime = time(&now);
        rs.ad.endtime = rs.ad.starttime + 60 * 60 * 24 * 30;
    }
    else {
        localTOUCT(rs.ad.starttime);
        localTOUCT(rs.ad.endtime);
    }

    if( rs.isNull(12) ) {
        // ad server needs fake times for now...
        if( fortargeting ) {
            time_t now;
            rs.ad.starttime = time(&now) - 60 * 60 * 24 * 15;
            rs.ad.endtime = now + 60 * 60 * 24 * 15;
        }
        else {
            rs.ad.starttime = rs.ad.endtime = 0;
        }
    }
    else {
        localTOUCT(rs.ad.starttime);
        localTOUCT(rs.ad.endtime);
    }
}

```

Page 3 (8)

19-Jan-1996 10:15

SQLDB.CPP

```

static void makeDefaultAds(ADArray* ads)
{
    if( stream default("c:\lan\default_ads.txt");
    if( !default_ads_open() ) {
        ASSERT(FALSE);
        return;
    }

    message("db connection failed, using default_ads.txt");
    default_ads_mode = TRUE;

    while( 1 ) {
        char init[256];
        char jump[128];
        *fn = 0;
        defAd >> fn >> jumpTo;
        if( *fn == 0 )
            break;

        Ads ad = *new Ad;
        defaultAd = ad;
        time_t now;
        ad.starttime = time(&now) - 60 * 60 * 24 * 15;
        ad.endtime = now + 60 * 60 * 24 * 15;
        ad.filename = fn;
        ad.jumpTo = jumpTo;
        ads.Add(ad);
    }

    // 0=all
    // if fortargeting, update Ad::targetsite to reflect
    // site exclusions
    // active only
    // include where enddate has past or where all delivered
    // (for management and reporting...)
    // order from newest to oldest
    // exclude ads the specified site has approved

    DMOPD newestFirst, // exclude ads the specified site has approved
    DMOPD approvesiteID)

    {
        // calc time zone adjustment
        CTime t = CTime::GetCurrentTime();
        tm gmt, local;
        t.GetGmtTm(&gmt);
        t.GetLocalTm(&local);
        if( local.tm_hour > gmt.tm_hour )
            gmt.tm_hour += 24;
        utof = (gmt.tm_hour - local.tm_hour) * 60 * 60;

        ads.SetSize(0, 64);

        DMOPD active = 1;
        getConfigValue("Active", active);
        Ad* ad;
        char sql[2000];
        *select id, type, os, browser, domainType, ip, filename, jumpTo, frequency, image, series, \
        max_impressions, n_shown, datediff(ss, '1/1/70', starttime), datediff(ss, '1/1/70', endtime), \
        flags, hours_of_day, days_of_week, employees, sales, active, description, max_amount, po_number, \
        approved, n_jumps from placements"

        BOOL where = FALSE;

        if( !includeExpired ) {
            strcat(sql, " where (max_impressions=0 or n_shown<max_impressions) and \
            (end_time=null or end_time>getdate())");
            where = TRUE;
        }

        if( !activeOnly ) {
            if( where ) {
                strcat(sql, " and");
            }
            else {
                ...
            }
        }
    }
}

```

HIGHLY

19-Jan-1996 10:15

SOLDB.CPP

```

// load pages to include/exclude
for( i = 0; i < ads.GetSize(); i++ ) {
    Ad ad = *ads.GetAt(i);
    if( !ad.IsTargeted() )
        continue;
    DWORD pageID;
    BOOL include;
    Cursor c;
    c.Bind( SQL_C_LONG, pageID, sizeof(pageID) );
    c.Bind( SQL_C_LONG, include, sizeof(include) );
    char sql[512] = "select page_id,include from placement_pages where ad_id=";
    addValue(sql, ad.id, FALSE);
    c.exec(sql);
    int n = 0;
    while( c.FetchNext() ) {
        if( ad.targetPages.IsEmpty() ) {
            ad.targetPages.InitHashTable(37);
            ad.includePages = include;
        }
        ad.targetPages.SetAt( pageID, TRUE );
        n++;
    }
    if( n > 31 ) {
        message("Increase Ad::targetPages hash size");
    }
}

// load site/page categories
for( i = 0; i < ads.GetSize(); i++ ) {
    Ad ad = *ads.GetAt(i);
    if( !ad.IsTargeted() )
        continue;
    DWORD siteID;
    Cursor c;
    c.Bind( SQL_C_LONG, siteID, sizeof(siteID) );
    char sql[512] = "select interest_id from placement_sitecats where ad_id=";
    addValue(sql, ad.id, FALSE);
    c.exec(sql);
    int n = 0;
    while( c.FetchNext() ) {
        if( ad.siteCategories.IsEmpty() ) {
            ad.siteCategories.InitHashTable(37);
            ad.siteCategories.SetAt( interestID, TRUE );
            n++;
        }
        if( n > 31 ) {
            message("Increase Ad::siteCategories hash size");
        }
    }

    // load sites
    for( i = 0; i < ads.GetSize(); i++ ) {
        Ad ad = *ads.GetAt(i);
        if( !ad.IsTargeted() )
            continue;
        int n = 0;
        Cursor c;
        c.Bind( SQL_C_LONG, sn, sizeof(n) );
        char sql[512] = "select count(*) from placement_sics where ad_id=";
        addValue(sql, ad.id, FALSE);
        c.exec(sql);
        if( c.FetchNext() )
            continue;
        if( n == 0 )
            continue;
        if( n > 100 )
            message("=>100 SICS targeted");
    }
}
Cursor c;
CString site;
c.Bind(sqlc);

```

Page 5(8)

19-Jan-1996 10:15

SOLDB.CPP

```

Ad ad = new Ad(rs.ad);
ad->calcSite();
if( ad->id == cbaKeyAdID && !ad.IsTargeting() ) {
    delete badKeyErrorAd;
    badKeyErrorAd = ad;
}
else {
    ad.Add(ad);
    if( defaultAd == 0 && ad->type != Ad::Type && !ad.IsTargeted() )
        defaultAd = ad;
}

if( main.Commit() ) {
    // load sites to include/exclude
    for( int i = 0; i < ads.GetSize(); i++ ) {
        Ad ad = *ads.GetAt(i);
        if( !ad.IsTargeted() )
            continue;
        DWORD siteID;
        BOOL include;
        Cursor c;
        c.Bind( SQL_C_LONG, siteID, sizeof(siteID) );
        c.Bind( SQL_C_LONG, include, sizeof(include) );
        char sql[512] = "select site_id,include from placement_sites where ad_id=";
        addValue(sql, ad.id, FALSE);
        c.exec(sql);
        int n = 0;
        while( c.FetchNext() ) {
            if( ad.targetSites.IsEmpty() ) {
                ad.targetSites.InitHashTable(37);
                ad.includeSites = include;
            }
            ad.targetSites.SetAt( siteID, TRUE );
            n++;
        }
        if( n > 31 ) {
            message("Increase Ad::targetSites hash size");
        }
    }

    if( !ad.IsTargeting() ) {
        // Load site exclusions of placements. If exclude this ad,
        // and Ad::includeSites is TRUE, remove site from map. If
        // exclude this ad, and Ad::includeSites is FALSE, add this
        // site to the map.
        for( int i = 0; i < ads.GetSize(); i++ ) {
            Ad ad = *ads.GetAt(i);
            DWORD siteID;
            Cursor c;
            c.Bind( SQL_C_LONG, siteID, sizeof(siteID) );
            char sql[512] = "select site_id from placement_banned where ad_id=";
            addValue(sql, ad.id, FALSE);
            c.exec(sql);
            while( c.FetchNext() ) {
                if( ad.targetSites.IsEmpty() ) {
                    ad.targetSites.InitHashTable(37);
                    ad.includeSites = FALSE; // exclude
                }
                if( ad.includeSites ) {
                    ad.targetSites.RemoveKey(siteID);
                    if( ad.targetSites.GetCount() == 0 ) {
                        // since map is empty, will go to all sites,
                        // which is wrong. Deactivate.
                        ad.starttime = ad.endtime = 0;
                        message( CString("error, no sites allowed for ") + ad.FileName );
                    }
                }
                else
                    ad.targetSites.SetAt(siteID, TRUE);
            }
        }
    }
}

```

HIGHLY  
CONFIDENTIAL

DC 069510

19-Jan-1996 10:15

SQLDB.CPP

```

if( ads.GetSize() == 0 && !forTargeting ) {
    // db connection down, use some default ads
    makeDefaultAds(ads);
}

if( defaultAd == 0 ) {
    TRACE("no default ad\n");
    message("no default ad");
}

return ads.GetSize() != 0 && defaultAd != 0;
}

```

Page 7(8)

19-Jan-1996 10:15

SQLDB.CPP

```

char sql[512] = "select siccode from placement_sic where ad_id=";
addValue(sql, ad.id, FALSE);
c.exec(sql);
SICCode = 0;
while( c.fetchNext() ) {
    stripSpaces(sql);
    if( s == 0 ) {
        // to do: count the # of SICs (first, and allocate that number
        // rather than 50
        s = new SICCode[n];
        ad.sicCodes = s;
    }
    s = sic;
    if( ++ad.sicCodes == n ) {
        ASSERT( !c.fetchNext() );
        break;
    }
    s++;
}

// load regional
for( i = 0; i < ads.GetSize(); i++ ) {
    Region r = 0;
    Ad ad = ads.GetAt(i);
    if( !ad.isTargeted() )
        continue;

    int n = 0;

    Cursor C;
    C.bind(SQL_C_LONG, &n, sizeof(n));
    char sql[512] = "select count(*) from placement_locations where ad_id=";
    addValue(sql, ad.id, FALSE);
    c.exec(sql);
    if( !c.fetchNext() )
        continue;
    if( n == 0 )
        continue;
    if( n > 100 )
        message("=>100 locations targeted");
}

Cursor C;
WORD country;
CString state, zip;
int areaCode;
C.bind(SQL_C_LONG, &country, sizeof(country));
C.bind(&state);
C.bind(&zip);
C.bind(SQL_C_LONG, &areaCode, sizeof(areaCode));
char sql[512] = "select country,state,zipcode,areacode from placement_locations where ad=";
addValue(sql, ad.id, FALSE);
c.exec(sql);
areaCode = 0;
while( c.fetchNext() ) {
    if( i == 0 ) {
        i = new Region(n);
        ad.locations = i;
    }
    i->country = country;
    i->state = state;
    i->zipCode = zip;
    i->areaCode = areaCode;
    if( ++ad.locations == n ) {
        ASSERT( !c.fetchNext() );
        break;
    }
    i++;
    areaCode = 0;
}

if( main.commit() )

```

HIGHLY  
CONFIDENTIAL

DC 069511



16-Jan-1996 14:03

SERVER.CPP

```

// server.cpp
//
#include <stdio.h>
#include <stream.h>
#include <server.h>
#include <toolkit/sock.h>
#include <toolkit/mapstate.h>
#include <toolkit/tsutil.h>
#include <defined_ADSVR>
#include <getrequest.h>
#include <tables.h>
old defined( IAP )
void message(const char *);
void defined( IAP )
#include <request.h>
void qpurge();
void message(const char *) { }
false
#include <request.h>
#include <mgmtrequest.h>
void message(const char *) { }
endif

#include <toolkit/crit.h>
extern CriticalSection fast;

const char cHTTPVer[] = "HTTP/1.0 ";
const char cContentLen[] = "Content-Length: ";
const char cErrHeader[] = "Error: ";
const char cErrTrailer[] = "<hr/>";
const char cContentHTML[] = "Content-Type: text/html";

extern int nListenThreads;

ofstream errLog;

void sendError(Connection *c, const char *msg, const char *headerField)
{
    char buf[10];
    CString s = cHTTPVer;
    s += msg;
    s += "\r\n";
    s += cContentHTML;
    if (headerField)
        s += headerField;
    s += cErrHeader;
    int len = strlen(msg) + strlen(cErrHeader) + strlen(cErrTrailer);
    int len = strlen(buf, 10);
    s += "\r\n\r\n";
    s += cErrHeader;
    s += msg;
    s += cErrTrailer;
    c->write( (const char *) s, s.GetLength() );
}

bool addressOK(const sockaddr_in from)
{
    if (from.sin_addr.s_un.b_1 == 206 &&
        from.sin_addr.s_un.b_2 == 4 &&
        from.sin_addr.s_un.b_3 == 219 )
    {
        // IAP network
        return TRUE;
    }
    return FALSE;
}

void serviceRequest(Connection *c, const sockaddr_in from)
{
    // if (addressOK(from) )
    //     return;

```

HIGHLY  
CONFIDENTIAL

DC 069512

SERVER.CPP

16-Jan-1996 14:03

Page 2 (4)

```

const BUFSIZE = 12768;
char buf(BUFSIZE);
int n = 0; // total n bytes read
const char *p = buf; // Content-length
int countDown = 0; // Content-length
Connection::ReadError err = Connection::OK;
while(1) {
    int toRead = BUFSIZE - n - 1;
    int nRead = c->read(buf + n, toRead, err);
    n += nRead;
    buf[n] = 0;
    if (countDown > 0) {
        countDown -= nRead;
        if (countDown <= 0)
            break;
    }
    if (nRead == 0) {
        // error
        break;
    }
    const char *p;
    if (p = strstr(buf, "\r\n\r\n")) {
        const char *cl = strstr(buf, "Content-length:");
        if (!cl)
            cl = strstr(buf, cContentLen);
        if (!cl) {
            cl = 15;
            sscanf(cl, "%ld", &countDown);
            countDown -= strlen(p + 4); // decrement by what we've already got
            countDown -= n - ((p + 4) - buf); // decrement by what we've already got
            if (countDown > 0)
                continue;
        }
        break;
    }
    Verb v = UNKNOWN;
    const char *r = buf;
    if (!strstr(buf, "get ", 4) == 0) {
        v = GET;
        r += 4;
    }
    else if (!strstr(buf, "head ", 5) == 0) {
        v = HEAD;
        r += 5;
    }
    else if (!strstr(buf, "post ", 5) == 0) {
        v = POST;
        r += 5;
    }
    if (v == UNKNOWN) {
        if (!buf == 0) {
            sendError(c, "400 bad request");
            if (buf[1] != 0) {
                message("empty request, buf:");
            }
            else if (err == Connection::Timeout) {
                message("empty request, timeout");
            }
            else if (err == Connection::ReadErr) {
                message("empty request, readerr");
            }
            else {
                message("empty request, err=OK");
            }
        }
        else {
            sendError(c, "501 Not Implemented");
        }
        return;
    }
}

```

SERVER.CPP

```

if defined_IAP
    IAPRequest gr(c, v, r, from);
    if defined_ADSVR
        GetRequest gr(c, v, r, from);
    else
        HttpRequest gr(c, v, r, from);
    sendif
        gr.service();
}

Listener *listener = 0;

static DWORD ad = GetTickCount();
static HANDLE hThread = 0;
int maxThreads = 1;

DWORD WINAPI Thread(LPVOID)
{
    static DWORD ad = GetTickCount();
    while(1) {
        Connection *c = listener->waitForConnection(from);
        if(c) {
            {
                Crit c(fast);
                int x = ++nThreads;
                if(x > maxThreads)
                    maxThreads = x;
            }
            serviceRequest(c, from);
            delete c;
        }
        {
            Crit c(fast);
            nThread--;
            if(nThread == 0) {
                // idle
                qpurge();
            }
        }
    }
    return 0;
}

BOOL startServer()
{
    if defined_ADSVR
        if(!openTables()) {
            MessageBox("Error opening tables");
            return FALSE;
        }
    if(!initMinsock()) {
        return FALSE;
    }
    mapStateInit();
    InitCountryTimezoneTable();
    sendif
    if 0
    {
        // TEMP!
        Connection c;
        if(c.connect("www.microsoft.com", 80)) {
            c.write("GET /adef HTTP/1.0\r\n\r\n", 22);
            while(1) {
                char buf[256];
                int n = c.read(buf, 255);
            }
        }
    }
}

```

HIGHLY  
CONFIDENTIAL

DC 069513

SERVER.CPP

```

if(n) {
    buf[n] = 0;
    TRACE("%s", buf);
}
else
    break;
}
return TRUE;
}
}

if defined_PORT
    int port = _PORT;
else
    int port = 80;
}

Listener *new Listener(port) {
    if(listener->ok()) {
        errLog.open("c:/an/errlog.txt",
            ios::out | ios::app,
            filebuf::sh_read);
        ASSERT(errLog.is_open());
        errLog << "-----ad server started\n"; errLog.flush();
    }
}

for(int i = 0; i < nListeners; i++) {
    Sleep(100); // [dam] this is a test; sometimes it doesn't listen right, just a hunch
    AfxbeginThread(listenerThread, 0);
}
}
else
    ASSERT(FALSE);
return TRUE;
}
}

```

29-Dec-1995 16:52

USERS.CPP

```

if( c.timedOut() ) {
    *timedOut = TRUE;
    delete u; u = 0;
}
else if( c.fetchNext() ) {
    u->userID = userID;
}
else {
    delete u;
    u = 0;
}
return u;
}

User* User::_lookupUserByAddress(Database& db, DWORD ip, BOOL *timedOut)
{
    User *u = new User;
    UserCursor c(db, u);
    c.minimalBind();
    c.bind( SQL_C_LONG, &u->userID, 4 );
    char sql[128]; *select ftp_tried, has_cookie, id from users where ip=ip";
    wprintf(sql, "select ftp_tried, has_cookie, id from users where ip=%d",
        (const char *) sqlipStr(ip));
    if( timedOut != 0 )
        c.setTimeout();
    c.exec(sql);

    if( c.timedOut() ) {
        *timedOut = TRUE;
        delete u;
        u = 0;
    }
    else if( !c.fetchNext() ) {
        delete u;
        u = 0;
    }
    return u;
}

void User::updateFTPTried(Database& db)
{
    if( !tempUserObject() ) {
        ASSERT(FALSE);
        return;
    }

    char buf[256]; *update users set ftp_tried=id where id=id";
    wprintf(buf, "update users set ftp_tried=%d where id=%d",
        ftpTried ? 1 : 0, userID);
    db.exec(buf);
    db.commit();

    void User::makePermanent(Database& db)
    {
        if( !tempUserObject() )
            return;
        ASSERT( name.isEmpty() && title.isEmpty() && emailAddr.isEmpty() );
        // add to DB
        char buf[1096] =
            "insert users (ip, browser, bver1, bver2, os, domain_type, is_proxy, is_networkdesc, ftp_tried, ha";
        strcat(buf, " values (");
        addBinValue(buf, ip);
        addValue(buf, browser);
        addValue(buf, bver1);
        addValue(buf, bver2);
        addValue(buf, os);
        addValue(buf, domainType);
        addBool(buf, proxy);
        addBool(buf, isNetworkDescription);
        addBool(buf, ftpTried);
    }
}

```

Page 1(3)

29-Dec-1995 16:52

USERS.CPP

```

// users.cpp
#include "stdafx.h"
#include "objects.h"
#include "d/cookiekit/db.h"
#include "d/cookiekit/inf_util.h"
#include "d/cookiekit/dbutil.h"

/* Implementation for hash tables
User* User::_lookupUserID(DWORD userID)
{
    User *u = new User;
    return u;
}

User* User::_lookupUserByAddress(DWORD ip)
{
    DWORD userID = networkNodeTable->getUserID(ip, FALSE);
    if( userID == 0 ) {
        // Try to get domain info at least. Note: if user is uniquely
        // identifiable, derive data process will create a record for the
        // user as soon as it gets a chance.
        userID = networkNodeTable->getUserID(justNetworkNumber(ip), TRUE);
    }
    if( userID ) {
        return _lookupUserID(userID);
    }
    return 0;
}

class UserCursor : public Cursor
{
public:
    UserCursor(Database& db, User *u) : Cursor(db),
        u(u) {}

    // just gets field that aren't derivable from request header
    void minimalBind()
    {
        bind( SQL_C_LONG, &u->ftpTried, sizeof(BOOL) );
        bind( SQL_C_LONG, &u->hasCookie, sizeof(BOOL) );
    }

    User *u;
};

void User::_lookupAncillaryInfo(Database& db)
{
    if( userID == 0 ) {
        return;
    }

    Cursor c(db);
    char sql[128]; *select email from users where id=id", userID);
    wprintf(sql, "select email from users where id=%d", userID);
    c.bind(emailAddr);
    c.exec(sql);
    c.fetchNext();
    db.commit();
}

User* User::_lookupUserID(Database& db, DWORD userID, BOOL *timedOut)
{
    User *u = new User;
    UserCursor c(db, u);
    c.minimalBind();
    char sql[128];
    wprintf(sql, "select ftp_tried, has_cookie from users where id=id", userID);
    if( timedOut != 0 )
        c.setTimeout();
    c.exec(sql);
}

```

HIGHLY  
CONFIDENTIAL

DC 069514

29-Dec-1995 16:52

USERS.CPP

```
addBool(buf, hasCookie, FALSE);
atreat(buf, *) =);
if (db.doInsert(buf) == 1) {
    Cursor c(db);
    c.bind(SOL_C_LONG, userID, 4);
    strcpy(buf, "select max(id) from users where ip=");
    addInValue(buf, ip, FALSE);
    c.exec(buf);
    c.fetchNext();
    ASSERT(userID != 0);
}
db.commit();
}
```

HIGHLY  
CONFIDENTIAL

DC 069515

20-Jan-1996 10:12

SITEPAGE.CPP

```

SITEPAGE.CPP
//
// Didn't find the page. Add page if site is correct.
{
    CString siteKey(from, q + from);
    int approved = 0;
    Cursor c(db);
    c.bind(SOL_C_LONG, sp->siteID, sizeof(sp->siteID));
    c.bind(SOL_C_LONG, approved, sizeof(approved));
    CString sql = "select id,approved from sites where keyname='";
    sql += siteKey + "'";
    c.exec(sql);
    if(c.fetchNext()) {
        if(approved == 0) {
            message(CString("unapproved site: ") + from);
        }
        else {
            delete p;
        }
    }
    else {
        delete p;
        p = 0;
        if(! defaultAdmMode)
            message(CString("unknown site: ") + from);
    }
}
return p;
}

void SitePage::add(Database db, const char *keyname)
{
    char buf[512] = "insert sitepages(junk, keyname, site, categorized) values('",";
    addValue(buf, keyname);
    addValue(buf, (int) siteID);
    addValue(buf, (int) categorized, FALSE);
    strcat(buf, "');");
    if(db.exec(buf) != 1) {
        TRACE("error adding sitekey\n");
        CString s = "sql: ";
        s += buf;
        ASSERT(FALSE);
        TRACE(s);
        message(s);
    }
}

Cursor c(db);
id = 0;
c.bind(SOL_C_LONG, sid, 4);
strcpy(buf, "select id from sitepages where keyname='");
addValue(buf, keyname, FALSE);
c.exec(buf);
if(c.fetchNext()) {
}
}

```

Page 1(2)

20-Jan-1996 10:12

SITEPAGE.CPP

```

// sitepage.cpp
//
#include "stdafx.h"
#include "object.h"
#include "dtoolkit/db.h"
#include "dtoolkit/iat_util.h"
#include "dtoolkit/dbutil.h"

void message(const char *s);

SitePage::SitePage()
{
    id = 0;
    siteID = 0;
    categorized = FALSE;
}

void SitePage::loadCategories()
{
    DWORD interestID;
    Cursor c;
    c.bind(SOL_C_LONG, interestID, sizeof(interestID));
    char sql[1024] = "select interests_id from page_categories where page_id='";
    addValue(sql, id, FALSE);
    strcat(sql, " union all select interests_id from site_categories where site_id='";
    addValue(sql, siteID, FALSE);
    c.exec(sql);
    while(c.fetchNext()) {
        categories.Add(interestID);
    }
}

extern BOOL defaultAdmMode;

SitePage::SitePage::lookupPage(Database db, const char *from, const char *requestHdr)
{
    // from key format: sitekey/docname
    if(from == 0)
        return 0;

    if(! strlen(from, "www.", 4) == 0)
        from += 4;

    if(from == 0)
        return 0;
    const char *q = strchr(from, '/');
    if(q == 0 || strlen(from) > 75)
        return 0;

    CString key;
    {
        // truncate a unique number from the end of the key
        const char *lastSlash = strchr(q + 1, '/');
        if(lastSlash == lastSlash) {
            key = CString(from, lastSlash - from);
        }
        else {
            key = from;
            if(key.GetLength() > 64)
                key = key.Left(64); // truncate to column width
        }
    }

    SitePage *p = new SitePage;
    {
        Cursor c(db);
        c.bind(SOL_C_LONG, sp->id, 4);
        c.bind(SOL_C_LONG, sp->siteID, 4);
        c.bind(SOL_C_LONG, sp->categorized, 4);
        char sql[1024] = "select id,site,categorized from sitepages where keyname='";
        addValue(sql, key, FALSE);
        c.exec(sql);
        if(c.fetchNext()) {
            return p;
        }
    }
}

```

HIGHLY  
CONFIDENTIAL

DC 069516

19-Jan-1996 15:58

AD.CPP

```
time_t t;
DWORD totalSpan = endTime - startTime;
if (totalSpan == 0)
    totalSpan = 1;
DWORD span = time(t) - startTime; if (span == 0) span = 1;

sl =
(DWORD) (((double) nShown /
(double) span / totalSpan) /
(maxImpressions) * 1000);

void Ad::show()
{
    nShown++;
    // if (nShown % 8 == 0) {
    //     // update sl
    //     calcSl();
    // }
    Ad::Ad()
    {
        daysOfWeek = 0x7f;
        started = FALSE;
        flags = Production | SpreadEvenly;
        sl = 1100;
        siCodes = 0;
        nSiCodes = 0;
        frequency = 0;
        imageSeries = FALSE;
        id = 0;
        maxImpressions = 0;
        nShown = 0;
        nJumps = 0;
        type = Normal;
        nLocations = 0;
        incLim = 0;
        gender = 0;
        maxAmount = 0;
        active = 0;
        approved = 0;
        includePages = 0;
        includeSites = 0;
        startTime = 0;
        endTime = 0;
        os = DefaultMask;
        browser = DefaultMask;
        domainType = DefaultMask;
        isp = DefaultMask;
        hoursOfDay = 0xffff;
        nEmployees = DefaultMask;
        salesVolume = DefaultMask;
        gender = DefaultMask;
        seriesNext = 0;
    }
    CString Ad::getFileName()
    {
        if (!imageSeries || seriesNext == 1)
            return fileName;

        char buf[256];
        sprintf(buf, "%ad-gif", (const char *) fileName.Left(fileName.GetLength() - 4), seriesNext);
        return buf;
    }
    CString Ad::fullName()
    {
        return gifRootDir + getFileName();
    }
    #if !defined(_ADSVF)

```

19-Jan-1996 15:58

AD.CPP

```
// ad.cpp
//
#include "atdafx.h"
#include "atctres.h"
#include "atstream.h"
#include "winsock.h"
#include "objects.h"
#include "d/cookie/ist_util.h"
#include "d/cookie/db.h"
#include "d/cookie/dbutil.h"
#include "d/derive/sqlderive.h"
#include "d/newderive/sig.h"
#include "rememberad.h"

const CString gifRootDir = "c:\\lan\\ads\\";

#if !defined(DERIVE)
int main() { return ads.GetSize(); }
#endif

extern Database lafmain;

//-----
// Ad
Ad::Ad()
{
    delete[] locations;
    delete[] siCodes;
}

Ad::Ad(const Ad& ad) :
    started(ad.started),
    id(ad.id), fileName(ad.fileName), jumpTo(ad.jumpTo),
    type(ad.type), os(ad.os), browser(ad.browser),
    domainType(ad.domainType), isp(ad.isp),
    maxImpressions(ad.maxImpressions), nShown(ad.nShown),
    nLocations(ad.nLocations), nSiCodes(ad.nSiCodes),
    frequency(ad.frequency), imageSeries(ad.imageSeries),
    seriesNext(ad.seriesNext), startTime(ad.startTime), endTime(ad.endTime),
    si(ad.si), flags(ad.flags), daysOfWeek(ad.daysOfWeek),
    hoursOfDay(ad.hoursOfDay), salesVolume(ad.salesVolume),
    nEmployees(ad.nEmployees), salesVolume(ad.salesVolume),
    gender(ad.gender), adDescription(ad.adDescription),
    maxAmount(ad.maxAmount), szPONumber(ad.szPONumber),
    active(ad.active), includeSites(ad.includeSites),
    includePages(ad.includePages), approved(ad.approved),
    nJumps(ad.nJumps)
{
    stripSpaces(fileName);
    stripSpaces(jumpTo);

    locations = 0;
    if (nLocations) {
        locations = new Region[nLocations];
        for (int i = 0; i < nLocations; i++) {
            locations[i] = ad.locations[i];
        }
    }

    siCodes = 0;
    if (nSiCodes) {
        siCodes = new siCode[nSiCodes];
        for (int i = 0; i < nSiCodes; i++) {
            siCodes[i] = ad.siCodes[i];
        }
    }

    void Ad::calcSl()
    {
        if (maxImpressions == 0)
            return;
    }
}

```

DC 069517  
HIGHLY  
CONFIDENTIAL



19-Jan-1996 15:58

AD.CPP

```

        brc = FALSE;
        break;
    }

    // Now save the SICs to the "placement_sics" table
    for (nloop = 0; nloop < nSICCodes; nloop++)
    {
        wsprintf( buf, "insert placement_sics(ad_id,siccode) values(td,'%s')",
            adID, sicCode);
        if ( !afmain.exec( buf ) != 1 )
        {
            ASSERT( 0 );
            brc = FALSE;
            break;
        }
    }

    // Now save the site categories to the placement_sitcats table
    POSITION pos = siteCategories.GetStartPosition();
    DWORD dwInterestID;
    BOOL bJunk;
    while (pos)
    {
        siteCategories.GetNextAssoc( pos, dwInterestID, bJunk );
        wsprintf( buf, "insert placement_sitcats(ad_id,interest_id) values(td,td)",
            adID, dwInterestID );
        if ( !afmain.exec( buf ) != 1 )
        {
            ASSERT( 0 );
            brc = FALSE;
            break;
        }
    }

    // Now save the user interests to the placement_interests table
    pos = interests.GetStartPosition();
    while (pos)
    {
        interests.GetNextAssoc( pos, dwInterestID, bJunk );
        wsprintf( buf, "insert placement_interests(ad_id,interest_id) values(td,td)",
            adID, dwInterestID );
        if ( !afmain.exec( buf ) != 1 )
        {
            ASSERT( 0 );
            brc = FALSE;
            break;
        }
    }

    // Now save site include-exclude list in the placement_sitcs table
    pos = targetSites.GetStartPosition();
    while (pos)
    {
        targetSites.GetNextAssoc( pos, dwSiteID, bJunk );
        wsprintf( buf, "insert placement_sitcs(ad_id,sitc_id,include) values(td,td,td)",
            adID, dwSiteID, includesSite );
        if ( !afmain.exec( buf ) != 1 )
    }

```

Page 5(9)

19-Jan-1996 15:58

AD.CPP

```

    {
        strftime( szTime, 9, "%m/%d/%y", gmtime( &startime ) );
        addValue( buf, szTime );
    }
    else
    {
        strcat( buf, "(null)," );
    }
}

strcat( buf, "end_time=" );
if ( !endTime )
{
    strftime( szTime, 9, "%m/%d/%y", gmtime( &endTime ) );
    addValue( buf, szTime, FALSE );
}
else
{
    strcat( buf, "(null)" );
}

strcat( buf, "where id=" );
addValue( buf, id, FALSE );

if ( !afmain.exec( buf ) != 1 )
{
    ASSERT( 0 );
    return( FALSE );
}

return( AddPlacementTables( id ) );
}

return( FALSE );
}

BOOL Ad::AddPlacementTables( DWORD adID )
{
    char buf[1024];
    BOOL brc = TRUE;
    while (TRUE)
    {
        // Now save the locations to the "placement_locations" table
        for (int nloop = 0; nloop < nLocations; nloop++)
        {
            strcpy( buf, "insert placement_locations(" );
            if ( !locations[nloop].country )
                strcat( buf, "country," );
            if ( !locations[nloop].state.IsEmpty() )
                strcat( buf, "state," );
            if ( !locations[nloop].zipCode.IsEmpty() )
                strcat( buf, "zipcode," );
            if ( !locations[nloop].areaCode )
                strcat( buf, "areacode," );
            strcat( buf, "ad_id) values(" );
            if ( !locations[nloop].country )
                addValue( buf, locations[nloop].country );
            if ( !locations[nloop].state.IsEmpty() )
                addValue( buf, locations[nloop].state );
            if ( !locations[nloop].zipCode.IsEmpty() )
                addValue( buf, locations[nloop].zipCode );
            if ( !locations[nloop].areaCode )
                addValue( buf, locations[nloop].areaCode );
            addValue( buf, adID, FALSE );
            strcat( buf, ")," );
        }
        if ( !afmain.exec( buf ) != 1 )
        {
            ASSERT( 0 );
        }
    }
}

```

HIGHLY  
CONFIDENTIAL

DC 069519



19-Jan-1996 15:58

AD.CPP

```

    ASSERT( 0 );
    brc = FALSE;
    break;
}

// Now save site page include-exclude list in the placement_sites table
// =====
pos = targetPages.GetStartPosition();
DWORD dwPageId;
while (pos)
{
    targetPages.GetNextAssoc( pos, dwPageId, blunk );
    wprintf( buf, "insert placement_pages(ad_id,page_id,include) values(%d,%d,%d)",
        adId, dwPageId, includePages );
    if ( !afmain.exec( buf ) != 1 )
    {
        ASSERT( 0 );
        brc = FALSE;
        break;
    }
    break;
}

afmain.commit();
return( brc );
}

BOOL Ad::Remove( BOOL bRemoveFromPlacements )
{
    char buf[1024];
    BOOL brc = TRUE;
    while (TRUE)
    {
        // Delete locations from the "placement_locations" table
        // =====
        wprintf( buf, "delete placement_locations where ad_id=%d", id );
        if ( !afmain.execErrorOK( buf ) != 0 )
        {
            ASSERT( 0 );
            brc = FALSE;
            break;
        }

        // Delete the SICs from the "placement_sics" table
        // =====
        wprintf( buf, "delete placement_sics where ad_id=%d", id );
        if ( !afmain.execErrorOK( buf ) != 0 )
        {
            ASSERT( 0 );
            brc = FALSE;
            break;
        }

        // Delete the site categories from the placement_sitcats table
        // =====
        wprintf( buf, "delete placement_sitcats where ad_id=%d", id );
        if ( !afmain.execErrorOK( buf ) != 0 )
        {
            ASSERT( 0 );
            brc = FALSE;
            break;
        }

        // Delete the user interests from the placement_interests table
        // =====

```

HIGHLY  
CONFIDENTIAL

DC 069520

19-Jan-1996 15:58

AD.CPP

```

// =====
wprintf( buf, "delete placement_interests where ad_id=%d", id );
if ( !afmain.execErrorOK( buf ) != 0 )
{
    ASSERT( 0 );
    brc = FALSE;
    break;
}

// Delete the site include-exclude list from the placement_sites table
// =====
wprintf( buf, "delete placement_sites where ad_id=%d", id );
if ( !afmain.execErrorOK( buf ) != 0 )
{
    ASSERT( 0 );
    brc = FALSE;
    break;
}

// Delete the site page include-exclude list from the placement_sites table
// =====
wprintf( buf, "delete placement_pages where ad_id=%d", id );
if ( !afmain.execErrorOK( buf ) != 0 )
{
    ASSERT( 0 );
    brc = FALSE;
    break;
}

if ( bRemoveFromPlacements )
{
    // =====
    // Lastly, delete the placement from the placements table
    // =====
    wprintf( buf, "delete placements where id = %d", id );
    if ( !afmain.execErrorOK( buf ) != 0 )
    {
        ASSERT( 0 );
        brc = FALSE;
        break;
    }
    break;
}

afmain.commit();
return( brc );
}

void Ad::Reset()
{
    daysOfWeek = 0x7f;
    flag = Production | SpreadEvenly;
    frequency = 0;
    imageSeries = FALSE;
    maxImpressions = 0;
    type = Normal;
    domainType = 0;
    gender = 0;
    maxAmount = 0;
    szPONumber.Empty();
    startTime = 0;
    endTime = 0;
    os = DefaultMask;
    browser = DefaultMask;
    domainType = DefaultMask;
    isp = DefaultMask;
    hoursOfDay = 0xffff;
    nEmployees = DefaultMask;
    salesVolume = DefaultMask;
    gender = DefaultMask;
    includePages = 0;
    includeSites = 0;
}

```

19-Jan-1996 15:58

AD.CPP

```
seriesNext = 0;
delete () siteCodes;
siteCodes = 0;
siteCodes = NULL;
delete () locations;
nLocations = 0;
locations = NULL;
targetPages.RemoveAll();
targetSites.RemoveAll();
siteCategories.RemoveAll();
interests.RemoveAll();
adDescription.Empty();
fileName.Empty();
jumpTo.Empty();
}
```

SendIt

**HIGHLY  
CONFIDENTIAL****DC 069521**